

# A Distributed Spatiotemporal Contingency Analysis for the Lebanese Power Grid

Fatima K. Abu Salem, Mohamad Jaber<sup>1</sup>, Chadi Abdallah<sup>2</sup>, Omar Mehio, and Sara Najem<sup>3</sup>

**Abstract**—We address a topological vulnerability analysis of the Lebanese power grid subject to random and cascading failures. Using an Apache Spark implementation that maps the topology of the grid to a complex network, we begin by developing a local structural understanding of the Lebanese power grid that reveals a certain level of decentralization via numerous connected components. Our Apache Spark implementation simulates the random and cascading sequences of events by which energy centers in Lebanon can be exposed and are at risk. The implementation is based on the bulk-synchronous parallel model and maintains optimal work, linear communication time, and a constant number of synchronization barriers. We complement this paper with a spatial understanding of the exposed hotspots. Our results reveal that failures in the power grid are spatially long-range correlated and correlations decay with distance. In a couple of attack scenarios, our Spark implementation achieves significant speedup on 16 cores for a graph with about  $9 \times 10^5$  nodes. Scalability toward 32 nodes improves when experimenting with replicas of the power grid graph which are double and quadruple the original size. This renders this paper suitable to larger networks at many vital levels beyond the power grid.

**Index Terms**—Graph, parallel computing, power grid, Spark, spatiotemporal.

## I. INTRODUCTION

IN SCALE-FREE networks (SFs), the probability of a node being connected to  $k$  others exhibits a power-law distribution  $P(k) \propto k^{-\alpha}$ , which is a topological property affecting and controlling their resilience or the measure of their functionality subject to disruptions [1]–[4]. Examples of this class of SF are the Internet, power systems, and transportation networks, which are real-world networks shown to be robust when subject to random failures, yet, displaying a high vulnerability when subjected to cascading ones [5]–[8].

In power systems, and unlike random failures that emerge locally, blackouts are severe events associated with a cascading behavior leading to global network collapse [9]–[15].

Manuscript received December 1, 2017; revised March 28, 2018 and July 24, 2018; accepted December 10, 2018. Date of publication January 16, 2019; date of current version February 12, 2019. This work was supported in part by the UNDP under Contract 00071806, in part by the Lebanese National Council for Scientific Research, and in part by the American University of Beirut. (Fatima K. Abu Salem, Mohamad Jaber, Chadi Abdallah, Omar Mehio, and Sara Najem contributed equally to this work.) (Corresponding author: Mohamad Jaber.)

F. K. Abu Salem, M. Jaber, and O. Mehio are with the Computer Science Department, American University of Beirut, Beirut 1107 2020, Lebanon (e-mail: fa07@aub.edu.lb; mj54@aub.edu.lb; okm02@aub.edu.lb).

C. Abdallah and S. Najem are with the National Center for Remote Sensing, National Council for Scientific Research, 1107 2260 Beirut, Lebanon (e-mail: snajem@cnrs.edu.lb; chadi@cnrs.edu.lb).

Digital Object Identifier 10.1109/TCSS.2018.2888689

Examples, such as the one affecting the north-east in the United States and Eastern Canada in 2003, burdened the respective economies with U.S. \$10 billion of direct costs [12]. Such failures can be linked to either structural dependences, where the damage spreads via structurally dependent connections, or functional overloads, where the flow goes through alternative paths leading to overloaded nodes. Thus, understanding the propagation of these failures becomes pivotal in developing and deploying protective and mitigating strategies.

Large power systems and real-world networks, in general, exhibit an exponentially increasing combinatorial number of failure nodes. Older works tackling contingency analysis relied on approximate power flow solutions, exhibiting only a simplified analysis of all combinatorial contingencies [16], [17]. These approaches also levy an overwhelming computational burden that cannot be accomplished in real time. Also, traditional approaches that employ the  $N - 1$  criterion are only able to investigate the ability of the transmission system to lose a power line or a power generator without causing an overload failure elsewhere. Examples of such approaches are employed by the North American Electric Reliability Corporation [18] and obviously do not capture the overload failures that propagate through interactions among a system's physical component. It thus becomes necessary to be able to perform an  $N - x$  ( $x \geq 2$ ) contingency analysis to assess whether a system can withstand the failure of any two or more components. In several leading works such as [4], [18], and [19], the power grid, particularly the network of its transmission lines, is analyzed using graph algorithms, such as betweenness centrality (BC) and shortest paths. In this paper, we build on the approach in [4] using an Apache Spark implementation of topological vulnerability analysis of the Lebanese power grid subject to random and cascading failures. Beyond the scholarly aspects of our proposed work, our analysis contributes toward precision-based policy making and disaster response in a region marred by wars and underdevelopment.

Using elements from the distributed algorithm design as well as spatial analysis, this paper extends the approach of [4] in analyzing the vulnerability of transmission lines to the assessment of that of the whole Lebanese power grid, including its generation, distribution, and transmission lines. However, in contrast to this cited body of work, our approach yields a system that is fault-tolerant to hardware failures and ensures the locality of reference to avoid data movement (e.g., network

communications in case of distributed setting, and garbage collector and memory allocation in case of local setting). Despite the fact that our input graph is not dramatically a “big graph,” our use of Spark is meant to provide for the following: 1) scalability, represented by a solution that is shown to remain promising and effective when the input graph scales to such daunting sizes, when tackling either of networks or power plants in larger countries; 2) fault tolerance, when the input graph size increases, failures—also known as faults—resulting from out-of-memory accesses and computations, message loss, and network error, to name a few causes, would begin to appear very frequently as opposed to rarely, which, in the absence of a fault-tolerant platform, would require one to reperform the whole computations. Spark provides for fault tolerance using lineage techniques as well as efficiently distributed in-memory computation. The choice of Spark also renders our system to be suitable for the clusters of commodity computers with relatively slow and cheap interconnects, and susceptible to many machine failures. Our implementation is based on the bulk-synchronous parallel (BSP) [20] model and employs the serial graph algorithms on distributed data in the single program, multiple data mode. Our data parallel algorithm maintains parallel optimal work, linear communication time, and a constant number of synchronization barriers.

This paper is organized as follows. In Section II, we present an overview of related work that tackles power grid resilience analysis as well as implementations of it that runs on distributed systems and describe some intrinsics related to Apache spark for big data distributed processing. In Section III, we present our distributed graph algorithm that builds on the loosely centralized structure of the Lebanese power grid using breadth-first search and vertex BC, orchestrated using four scenarios known in the literature [4]. Each of these scenarios simulates a unique temporal mode of removal of vertices. Despite the fact that our approach does not generalize to arbitrary networks, we believe that there is merit in tackling the specifics related to the Lebanese power grid, especially after Lebanon has withstood the decades of wars and a lack of infrastructure rebuilding. Still, our approach can generalize to networks of low connectivity such as that found in the Lebanese grid. In Section IV, we perform a run-time analysis of our algorithm and obtain excellent scalability for some scenarios as the number of processing cores increases up to 16, a result which we attribute to the large number of connected components found in the Lebanese power grid and the fact that the majority of connected components have a relatively small cardinality. In fact, scalability toward 32 nodes improves when experimenting with replicas of the power grid graph which are double and quadruple the original size. We demonstrate the loss in connectivity in the power grid associated with each scenario and obtain that the Lebanese power grid exhibits a relatively strong resilience thanks to its decentralized structure. We also perform a spatial correlation analysis of failures using the geocoding of vertices that lead to 90% loss in connectivity. Our results reveal that failures in the power grid are spatially long-range correlated and correlations decay with distance.

In Section V, we conclude with remarks on the impact of this paper.

## II. BACKGROUND

### A. Resilience Analysis Using Graph Theory

The seminal works in [4] and [19] address the contingency analysis for the power grid using the simulation of complex networks. Typically, the power grid is modeled using generator, transmission, and distribution nodes. In such SFs, a large fraction of the nodes have low degrees and a substantially smaller fraction of the nodes exhibit high degrees. Those networks are established to be resilient against random failures upon nodes, and the overall loss is to be contingent upon the targeting of the high-degree hubs [4], [21]–[23]. In addition to the topological analysis of the power grid, other metrics can be used to assess its vulnerability. These are based on the physical properties of the network such as line resistance and sensitivity, which encapsulate the impedance matrix, and consequently, the “electrical centrality,” the equivalent of topological centrality, is computed and node removal strategies can be tested accordingly [24], [25].

Equivalently, vertex vulnerability or network robustness can be evaluated through percolation measures such as the average inverse geodesic and the giant component, which emerges subsequent to attacks and exhibits a dependence on the node removal strategy and on the topology of the network under scrutiny [26]–[29]. We will thus focus our effort on a graph-based approach to vulnerability since the physical properties of the power grid were not made available to us.

In our Spark implementation, we aim to distribute the random and cascading failure scenarios explored in [4], which in turn models the North American power grid using its transmission lines and examines its connectivity in relation to a small set of high-impact nodes.

The notion of connectivity is based on the notion of BC. A node’s BC is a focal measure of connectedness in a graph, built around the notion of shortest paths. Given any two vertices in a graph, there exists at least one shortest path between the vertices. When the length of the path is infinity, it is understood to say that there is no path to connect the given two vertices. If the graph is weighted, the shortest path is obtained by minimizing the number of edges that the path passes through. Else, the path is obtained by minimizing the sum of the weights of the edges that the path passes through. Given an arbitrary vertex in a graph, its BC is defined to be the number of shortest paths that pass through the given vertex. Exploiting BC has been widespread in a number of works addressing power grid vulnerability analysis. The work in [4] employs this notion using four scenarios, each of which simulates a unique temporal mode of removal of vertices. For example, the overall connectivity of the graph is reexamined upon removal of transmission nodes according to the following orders: 1) totally random order; 2) decreasing order of node degrees (load); 3) decreasing order of node betweenness centrality; and 4) decreasing order of node BC in cascading order resulting from the nodes’ dynamical removal. The

cascading scenario is based on the recalculated information or more precisely after the identification of the most central node of the initial graph and subsequent to its removal a new graph ensues. The central node of the latter should then be computed, and then, the process is iterated over.

The approach taken in [18] considers the intensity of the power flowing on a transmission branch, as opposed to the degree of nodes. The resulting graph is weighted (but still undirected), and the contingency analysis method is based on applying edge BC [30] to the power grid topology. High-impact components in the power grid are defined using the most traversed edges. We believe that this approach is less exhaustive than the one adopted in [4] and do not pursue it here.

Our spatial understanding of the propagation of faults in the Lebanese power grid makes classical use of the concept of spatial correlation [31], [32]. As a leading example that we follow, this measure is used in [19] to measure the relation between the failures separated at some distance  $r$ . More details follow in Section III.

### B. Distributed Computation Frameworks

In the following, we discuss two distributed computation frameworks: 1) MapReduce and Pregel and 2) Spark and GraphX.

1) *MapReduce and Pregel*: The last few years have witnessed an uptake in distributed data processing research. The MapReduce paradigm [33] remains one of the leading frameworks for distributed computation on commodity hardware. A typical MapReduce program consists of the “Map” operator that parcels out work to various nodes within the cluster or map and the “Reduce” phase that applies a reduction operator on the results from each node into a global query. The key contributions of the MapReduce framework are the scalability and fault tolerance achieved for a variety of applications by optimizing the execution engine, for example, by reassigning tasks when a given execution fails. As with all other parallel and distributed paradigms, the performance of an efficient MapReduce algorithm is contingent upon a reduced communication cost. Of particular challenge is how to efficiently process large graphs. Graph algorithms often exhibit poor locality of reference and a low compute-to-memory access ratio, which affects the scalability of their parallel adaptations. It is also difficult to maintain a steady degree of parallelism over the course of execution of graph algorithms. Additionally, expressing a graph algorithm in MapReduce requires passing the entire state of the graph from one stage to the next, thus imposing significant communication as well as serialization in the parallel code.

The first serious development for supporting graph algorithms using the MapReduce framework is found in Google’s Pregel [34]. Instead of coordinating the steps of a chained MapReduce program, Pregel is able to process iteration over supersteps under the BSP model [20], [35], [36]. In a BSP algorithm, a computation proceeds in a series of global supersteps. Each superstep consists of three phases.

- 1) *A Concurrent Computation Superstep*: Each processor performs local computations using the values stored in the local, fast memory of the processor.
- 2) *A Communication Superstep*: The processes exchange the data between themselves if needed for the aggregation of the results computed in 1).
- 3) *A Barrier synchronization Superstep*: Each processor halts until all other processes have reached the same barrier.

According to this model, a graph algorithm in Pregel is organized as a sequence of iterations and can be described from the point of view of a vertex, which manages its state and sends messages only to its neighbors. Pregel keeps vertices and edges on the machine that performs computation and uses network transfers only for messages.

2) *Spark and GraphX*: Spark, a distributed computation framework built around the MapReduce paradigm, is a recent Apache foundation software project supported by an execution engine for big data processing. Spark provides for in-memory computation, which refers to the storage of information in the main random access memory (RAM) of dedicated servers rather than in relational databases running on relatively slower disk drives. Using over 80 high-level operators, Spark makes it possible to write code more succinctly and, till this point in time, is considered one of the fastest frameworks for big data processing. Spark’s most notable properties are also because of its core, which, in addition for serving as the base engine for large-scale parallel and distributed data processing, is able to handle memory management and fault recovery, scheduling, distributing, and monitoring jobs on a cluster, as well as interacting with storage systems.

Spark hinges on parallel abstract data collections called resilient distributed data sets (RDDs), which can be distributed across a cluster. These RDDs are immutable, partitioned data structures that can be manipulated through multiple operators, such as Map, Reduce, and Join. For example, RDDs are created through parallel transformations (e.g., map, group by, filter, join, and create from file systems). RDDs can be cached (in-memory) by allowing to keep data sets of interest locally across operations, thus contributing to a substantial speedup. At the same time, Spark uses lineage to support fault tolerance, i.e., record all the operations/transformations that yield RDDs from the source data. In case of failure, an RDD can be reconstructed, given the transformation functions contributing to that RDD. Additionally, after creating RDDs, it is possible to analyze them using actions, such as count, reduce, collect, and save. Note that all operations/transformations are lazy until one runs an action. At that point, the Spark execution engine pipelines operations and determines an execution plan.

Borrowing from Pregel, GraphX [37] is a platform built on top of Spark that provides APIs for parallel and distributed processing on large graphs. In GraphX, each graph is mapped into different RDDs, where in each RDD, one applies the computation on the graph using the “think like a vertex” model.

### III. MATERIALS AND METHODS

#### A. Input Graph

We build the network model using data provided by the Lebanese Ministry of Energy and Water. This data set contains the information about every power plant generator (sources for power), high-to-medium voltage transmission substations, medium-to-low voltage distribution stations that disseminate power directly to subscribers, as well as transmission lines through which power dissipates. Transmission lines exist between generators and transmission substations, as well as among transmission substations, distribution substations, and, finally, between transmission and distribution substations.

The power grid consists of generating stations responsible for the power production, which is transmitted through high-voltage lines to demand centers, which then, in turn, distribute power to the consumers. This design entails the emergence of a large number of connected components, which is a universal feature of grids with a characteristic power-law degree distribution [9]–[15]. There are five major power plants in Lebanon. Transmission and distribution substations are such that they receive power from all of the major power plants, and so, if all but one such plant is hit, the entire network can still receive power. Because of this redundancy, the only way to achieve total failure is through the obvious choice of attacking all five power plants. To exclude this obvious scenario from our contingency analysis, the data received directly from the Lebanese Ministry of Energy and Water does not include all five power plants from the associated graph model. This renders our contingency analysis completely focused on the irredundant nodes constituting transmission and distribution substations, where the corresponding graph representation of the power grid consists of several connected components.

We should note that in the original data set, these five vertices were assigned a tag directly by the Ministry of Energy and Water, which described them as redundant. Therefore, no extra computational work was done to identify them. In the event, when the power network fails to manifest a large number of connected components that permit for the kind of distribution we are adopting in this paper, one can attempt to distribute/parallelize the low-level graph BC and single-source shortest paths (SSSP) algorithms employed, using, for example, a number of distributed and parallel tools available in [38]–[44], to cite a few. Our analysis simulates the attacks on both transmission and distribution substations. It also pursues interdependences as failures propagate within a single-connected component and addresses the overall loss of connectivity to all subscribers as a result. Finally, our analysis adopts the idealized (and simplified) view that capacity across the transmission lines is never jeopardized, and thus, the failure of the network is a result of attacks on substations only.

Our resulting network model consists of an undirected, unweighted graph<sup>1</sup> with 679 965 edges, representing trans-

<sup>1</sup>Please note that this bears no impact on the performance if the graph is directed. First, all of the algorithms used (e.g., strongly connected component and BC) have variants that can tackle directed graphs. Moreover, these variants have the same work complexity as those for the undirected graph.

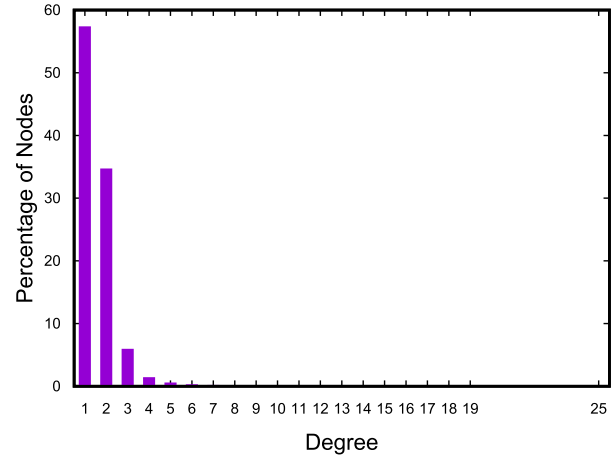


Fig. 1. Degree of vertices.

---

#### Algorithm 1: Random and Cascading Attacks

---

**Input:** Graph  $graph = (V, E)$   
**Output:** random and cascading attacks of the input graph

```

1 Function attackGraph (graph) :
2   for  $i \leftarrow 0$  to  $|V|$  do
3     select victim vertex  $v$ ;
4     remove vertex  $v$ ;
5     update loss with respect to  $v$ ;
6   end
7 End Function
    
```

---

mission/distribution lines, and 899 162 vertices, representing transmission/distribution substations. This is one order of magnitude larger than the graph treated in [4] and a consequence of the fact that the given Lebanese power grid representation captures extremely fine-grained spatial coordinates, yielding all distribution substations no matter how minor. Our distribution analysis of the degrees of vertices confirms that, indeed, the resulting graph exhibits a power-law distribution (see Fig. 1).

#### B. Random and Cascading Contingency Analysis

Our baseline approach for navigating through node attacks follows that of [4]. A high-level representation of the algorithm for implementing the random and cascading attacks is depicted in Algorithm 1. It captures both the notions of node attack, followed by an evaluation of the loss in power flow capacity across the entire network.

Here, a victim vertex is chosen upon an examination of some measure of its centrality. Centrality is a quantitative measure that aims at revealing the importance of a node. Several indices of centrality tackled in the literature are based on geometric, spectral, as well as path-based measures. We refer the reader to [45] for a comprehensive survey. For undirected graphs, the geometric measure related to the indegree of a given node and the path-based measure based on its betweenness have yielded a contextual understanding of connectivity loss in power grids [4], [18]. In line with this

**Algorithm 2: Scenario-Based Victim Selection**


---

**Input:** Graph  $graph = (V, E)$ , boolean cascading

- 1 Compute betweenness centrality for each  $v$  in  $V$ ;
- 2 **Function** `attackGraph( $graph, cascading$ )`:
- 3   **for**  $i \leftarrow 0$  **to**  $|V|$  **do**
- 4     select victim vertex  $v$  in  $V$ ;
- 5     remove vertex  $v$ ;
- 6     **if**  $cascading$  **then**
- 7       **for each node**  $u$  **not removed so far** **do**
- 8         recompute betweenness centrality for  $u$ ;
- 9       **end**
- 10    **end**
- 11    update loss with respect to  $v$ ;
- 12 **end**
- 13 **End Function**

---

body of work, we select the victim vertex according to one of the following four scenarios.

- 1) *Random Attacks*: A vertex is selected at random.
- 2) *Attacks Based on Geometric Centrality Measure*: The vertex with the highest degree is selected.
- 3) *Attacks Based on Path-Based Centrality Measure*: The vertex with the highest BC is selected. The BC of the nodes is only computed once (i.e., is not being updated after removing a vertex).
- 4) *Cascading Attacks*: This is similar to the BC scenario; however, the BC of the nodes is updated at each iteration (i.e., after removal of a victim vertex).

Algorithm 2 depicts that a more detailed snippet captures these four scenarios.

The choice behind node BC is motivated by the following. Let  $\sigma(s, t)$  denote the number of shortest paths between two vertices  $s$  and  $t$  and  $\sigma(s, t|v)$  denote the number of shortest paths between  $s$  and  $t$  that pass through  $v$ . A most recent BC score of  $v$  is suggested by Brandes [46] as follows:

$$C(v) = \sum_{s, t \in V} \frac{\sigma(s, t|v)}{\sigma(s, t)} \quad (1)$$

where this assumes  $s \neq t \neq v$  and considers  $\sigma(s, s) = 1$ ,  $\sigma(s, t|v) = 0$  if  $v = s$  or  $v = t$  and  $0/0 = 0$ . With this definition, a high centrality score  $C(v)$  implies that a vertex can reach others on relatively short paths or a vertex lies on a large number of shortest paths connecting other vertices. This particular choice of index is the basis for a quadratic running time, linear space algorithm by Brandes [46], which improves on former cubic running time algorithms. The algorithm follows an accumulation technique that invokes an SSSP algorithm in multiple iterations starting from each vertex in the graph whose score needs to be produced. When the graph is unweighted, SSSP can be solved using the breadth first traversals. To illustrate, let

$$\gamma(s, t|v) = \frac{\sigma(s, t|v)}{\sigma(s, t)}$$

denote the dependence of  $s$  and  $t$  on  $v$ , captured by the ratio of shortest paths between  $s$  and  $t$  that go through  $v$ . Also, let

$$\gamma(s|v) = \sum_{t \in V} \gamma(s, t|v)$$

denote the dependence of  $s$  on  $v$ , which is the sum of dependences of  $s$  and  $t$  on  $v$  for all possible targets  $t$ . We then have the following main result:

$$\gamma(s|v) = \sum_{w: (v, w) \in E \wedge d(s, w) = d(s, v) + 1} \frac{\sigma(s, v) \times (1 + \gamma(s|w))}{\sigma(s, w)}$$

where  $d(s, w)$  is the shortest path length from  $s$  to  $w$  [46]. The resulting algorithm now revolves around two steps.

- 1) For each  $s \in V$ , perform a breadth first traversal of the graph. In each traversal, compute the number of shortest paths from  $s$  to all  $t \in V$  going through a given other node  $v \in V$ .
- 2) After each traversal, accumulate  $\gamma(s, t|v)$  into  $\gamma(s|v)$ .

### C. Connected Components: A Localized View

In contrast to the relatively smaller graphs treated in the literature on power grid resilience analysis, we have maintained the bulk of the power grid down to the finest spatial coordinates. Our graph with  $|V| \approx 9 \times 10^6$  represents a challenging size for serial computers and opportunities for distributed algorithm design ought to be explored. The fact that the high-voltage power plants have been removed from the grid representation renders the graph a disconnected one, consisting of 199004 connected components, where each component consists of transmission/distribution substations as nodes, and transmission lines between them as edges. Our Spark algorithm will distribute the work over these components, and for this, we begin by explaining the contextual implications of a decentralized version of the grid as such. This distribution of the network leads us to readdress the connectivity/loss and BC in a manner that explores the effects of local changes on the global graph properties. We address those two issues separately in the following.

1) *Local Versus Global Connectivity Loss*: In [4], the notion of loss manifests itself when less and less paths connect a power plant to a distribution station. In contrast, the Lebanese power grid has adapted to decades of power cuts that have resulted in a significant dependence on diesel power generators which, in theory, can connect to any transmission substation, say, within districts, or distribution substations, at the level of neighborhoods. These diesel generators are mobile and, therefore, their location may remain “unknown” to an attacker, and they can be reasonably easily replaced. As such, the notion of loss in our scenario associates with attacks on transmission/distribution substations and the removal of transmission lines between them, excluding the status of power plant generators or diesel generators. A node retains a power supply in the network so long as there exists a path of edges connecting it to any other node. Each time a victim vertex is attacked, all of its incident edges are removed along with it. As a result, we adopt the following definitions for connectivity

and its loss. Given an undirected graph  $G = (V, E)$ , we define its connectivity as follows:

$$\text{connectivity}(G) = \sum_{v \in V} |\text{reachable}(v)|$$

where  $\text{reachable}(v)$  is the set of all reachable nodes from  $v$  via some path. This notion can be expanded as follows, where  $G$  denotes an undirected graph:

$$\begin{aligned} \text{connectivity}(G) &= |V| \times |V - 1| \quad \text{if } G \text{ is connected} \\ &= \sum_{i \in \{1, \dots, n\}} \text{connectivity}(G_i) \quad \text{otherwise.} \end{aligned} \quad (2)$$

The following formula captures the percentage of loss as a result of attacking (removing) a node  $x$  from  $G$ :

$$\text{loss}(G, x) = 1 - \frac{\text{connectivity}(G \setminus \{x\})}{\text{connectivity}(G)}$$

where  $G \setminus \{x\}$  is a graph defined by removing vertex  $x$  in  $G$  and all of its incident edges. For simplicity, we will track only the numerator appearing in this definition and consider hereafter that  $\text{loss}(G, x) = \text{connectivity}(G) - \text{connectivity}(G \setminus \{x\})$ . Proposition 1 reveals how the local loss within a component translates to global loss.

*Proposition 1:* Let  $G$  denote an undirected graph and  $G_i$  denote the connected component to which  $x$  belonged prior to its removal from  $G$ . We then have

$$\begin{aligned} \text{connectivity}(G) - \text{connectivity}(G \setminus \{x\}) \\ = \text{connectivity}(G_i) - \text{connectivity}(G_i \setminus \{x\}) \end{aligned} \quad (3)$$

where  $\text{connectivity}(G_i \setminus \{x\})$  can be computed as in (2).

*Proof:* Since  $G_i$  is a connected component of  $G$  and  $x \in G_i$ , we have

$$\begin{aligned} \text{connectivity}(G \setminus \{x\}) \\ = \text{connectivity}(G_i \setminus \{x\}) + \text{connectivity}(G \setminus G_i) \\ = \text{connectivity}(G_i \setminus \{x\}) + \sum_{j=1, j \neq i}^n \text{connectivity}(G_j). \end{aligned}$$

We now use this equation in

$$\begin{aligned} \text{connectivity}(G) - \text{connectivity}(G \setminus \{x\}) \\ = \sum_{j=1}^n \text{connectivity}(G_j) \\ - \left( \text{connectivity}(G_i \setminus \{x\}) \right. \\ \left. + \sum_{j=1, j \neq i}^n \text{connectivity}(G_j) \right) \\ = \text{connectivity}(G_i) - \text{connectivity}(G_i \setminus \{x\}). \end{aligned}$$

□

2) *Local Versus Global Centrality:* We now express the relationship between the global and local centrality measures for a given node within its connected component. We begin with the measure denoting the degree of a given vertex.

*Proposition 2:* Let  $\text{CC}(G) = \{G_1, \dots, G_n\}$  denote the connected components of the undirected power grid graph  $G$ . Let  $\text{deg}_G(v)$  denote the degree of a node  $v$  in  $G$  and  $\text{deg}_{G_i}(v)$  denote its degree in  $G_i$ , for some connected component  $G_i \in \text{CC}(G)$  containing  $v$ . We then have

$$\text{deg}_G(v) = \text{deg}_{G_i}(v).$$

*Proof:* The proof is straightforward. As  $G$  is undirected, the edges incident on  $v$  all belong to its connected component, and no edge outside its component can be incident on  $v$ . □

We now address the BC measure.

*Proposition 3:* Let  $\text{CC}(G) = \{G_1, \dots, G_n\}$  denote the connected components of the undirected power grid graph  $G$ . Let  $C_G(v)$  denote the BC score of a node  $v$  in  $G$  and  $C_{G_i}(v)$  denote its BC score in  $G_i$ , for some connected component  $G_i \in \text{CC}(G)$  containing  $v$ . We then have

$$C_G(v) = C_{G_i}(v).$$

*Proof:* Let  $s, t$ , and  $v$  denote distinct vertices in  $V$ . Since  $G$  is undirected, we have  $G_i \cap G_j = \emptyset, \forall i \neq j$ . Hence, if  $\exists i$  such that  $s, t \in G_i$ , then  $\sigma(s, t)$ , representing the number of paths between  $s$  and  $t$  in  $G$ , is identical to the number of paths between  $s$  and  $t$  in the subgraph  $G_i$ . Otherwise,  $\sigma(s, t) = 0$ . Similarly, if  $\exists i$  such that  $s, t, v \in G_i$ , then  $\sigma(s, t|v)$ , representing the number of paths between  $s$  and  $t$  in  $G$  that pass through  $v$ , is identical to the number of paths between  $s$  and  $t$  in the subgraph  $G_i$  that pass through  $v$ . Otherwise,  $\sigma(s, t|v) = 0$ . Using (1) for the BC score, we obtain that  $C_G(v) = C_{G_i}(v)$  in all cases. □

#### D. Spark-Based Implementation

We now present our Spark-based algorithm, guided by Propositions 1–3. The many connected components of the power grid provide for an inherent data distribution of the graph which, as we will see in Section IV, allows for a balanced workload as well as locality of reference. Spark employs a storage abstraction called RDDs. Each RDD is a distributed, fault-tolerant vector on which we can perform a set of vectorised operations. One can define a data partitioning scheme on an RDD. The execution engine can coschedule tasks on those RDDs to avoid data movement. In what follows, we refer to threads that may be referenced within one single multithreaded machine or across several machines.

- 1) Given a file (read from local or distributed file system, i.e., HDFS) containing information about the graph, build the corresponding graph RDD. Here, the input file is partitioned for distribution over the multiple threads. GraphX in Spark represents the graph internally as a pair of vertex and edge collections built on RDDs.
- 2) Compute the connected components on the graph RDD using GraphX's built in the strongly connected component distributed kernel. This creates an RDD, `rddCC`, of items, where each item corresponds to a connected component.

- 3) Partition  $\text{rddCC}$  into several  $p$  partitions, for  $p = 4, \dots, 32$ ,  $p \leftarrow 2 \times p$ . Each partition is allocated to a single thread and contains a number of connected components. Note that, we use a customized partitioning scheme to get balanced partitions, i.e., large strongly connected components are spread among different partitions.
- 4) The connected components within each partition are now distributed over the multiple threads on a single core (or multiple cores on a single thread). For each connected component, a thread  $t$  selects a victim node according to one of the four scenarios and locally updates the corresponding component by removing the victim vertex and its incident edges. Only the fourth scenario stipulates an update on the BC score of each remaining node. A tuple  $(\rho, \lambda)$  is appended to the end of a list  $\mathbf{L}_t$ . Here,  $\rho$  denotes its rank in the removal process if the vertex has been chosen in random order. Otherwise,  $\rho$  denotes its centrality (degree or load-based). Also,  $\lambda$  denotes the loss percentage associated with the given victim node.
- 5) Repeat Step 4 until all the nodes of a thread's connected component are identified.
- 6) All threads now communicate their pairs in the list  $\mathbf{L}_t$  to each other.
- 7) In the case where vertices have been removed in random order, define a reduce operation that concatenates the generated lists into a unified list  $\mathbf{L}$ . Else, define a reduce operation that merges all the generated lists on  $b$  into  $\mathbf{L}$ . This step is performed serially.
- 8) Simulate the attack on  $G$  by removing the nodes in  $\mathbf{L}$  starting from the head. For each node  $x_i$  removed from  $\mathbf{L}$ , return  $\text{loss}(G, x_i) = \sum_{j=1}^i \lambda_j$ . This step is also performed serially.

We now have the following.

*Proposition 4:* The above-mentioned algorithm is correct and returns  $\text{loss}(G \setminus \mathbf{L})$ , where  $\mathbf{L}$  denotes the list of all nodes attacked in  $G$ .

*Proof:* We first address Step 4. By Propositions 2 and 3, each local measure of a centrality by one thread is also the global measure, and as such, computing it locally yields the answer independently of the other connected components within the same thread or on other threads. Also, by a direct result of Proposition 3, updating the BC scores after reach removal in the cascading scenario requires information about the paths connecting each vertex to vertices only in its own local component and thus can be performed independently of other threads. This allows to parallelize the computation of all betweenness centralities after attacking nodes in the cascading scenario. At the end of Step 4, the loss upon removal of each vertex is computed locally on each connected component item. By Proposition 1, this local loss computed independently of other threads also indicates the global loss.

We now address Step 7. If the scenario applied is the random hit scenario, concatenating the vertices from each thread violates no specific ordering on them, and thus, one can proceed with the attacks as indicated by the concatenation. If the scenario applied relies on some measure of centrality, the merge procedure ensures that the locally ordered lists of

vertices produced by each thread now yield a totally ordered sorted list on all the centrality measures, after which the attacks on nodes can begin to take place.

We conclude with Step 8. Write  $\mathbf{L} = \{x_1, \dots, x_n\}$ , where  $n = |\mathbf{L}|$ . Let  $x$  and  $y$  denote any two arbitrary nodes of  $G$ , and assume, without loss of generality, that  $x$  was removed prior to  $y$ . Let  $\text{loss}(G \setminus \{x, y\})$  denote the loss in the graph after removing  $x$  followed by  $y$ . The losses returned in Step 7 are thus captured by  $\text{loss}(G \setminus \{x_1, \dots, x_n\})$ .

- 1) Suppose that  $x$  and  $y$  do not belong to the same connected component. This can happen either when both of them are tackled by the same thread or otherwise. Whether or not the failures of  $x$  and  $y$  are happening simultaneously, the loss incumbent on removing  $x$  is independent of the corresponding loss due to  $y$ , and so,  $\text{loss}(G \setminus \{x, y\}) = \text{loss}(G \setminus \{x\}) + \text{loss}(G \setminus \{y\})$ , i.e., the reduction operator on the individual losses can take place after the computation of individual losses locally in Step 4.
- 2) Otherwise, suppose that  $x$  and  $y$  belong to the same connected component. Then, they certainly are tackled by the same thread. In this case,  $x$  is failed before  $y$  is failed, and so  $\text{loss}(G \setminus \{x, y\})$  is computed correctly using the formula  $\text{loss}(G \setminus \{x\}) + \text{loss}((G \setminus \{x\}) \setminus \{y\})$ , which has already been computed locally in Step 4.  $\square$

We are now ready to conclude the analysis of our parallel design and follow the exposition in [47] and [48] to define the following terms. We call a parallel algorithm *work-optimal* if it has the smallest possible work, defined to be the smallest possible sequential work divided by the total number of processors. This excludes the cost of partitioning, which occurs at the beginning and thus is counted toward preprocessing costs, as well as the resulting data shuffling, which we regard as being memory or communication operations, not computation. We call a parallel algorithm *work-time optimal*, if in addition to being work-optimal, its time is best possible. In Proposition 5, we show that our resulting parallel design is work-optimal according to the above-mentioned definition. However, due to partitioning and data shuffling costs, our algorithm fails to be work-time optimal.

*Proposition 5:* The parallelized block in Steps 1–6 of the above-mentioned algorithm maintains optimal parallel work and requires linear time communication costs and a constant number of synchronization barriers in the BSP model. Particularly, let  $p$  denote the number of threads,  $W_s$  denote the serial work of the algorithm in Steps 1–6,  $g$  denote the machine-dependent cost in flops to communicate one data work, and  $\ell$  denote the machine-dependent cost in flops for all threads to synchronize. We then have

$$T_p = \Theta\left(\frac{W_s}{p}\right) + \Theta\left(\frac{|V|}{p}\right) + 2\ell \text{ flops.}$$

*Proof:* Here, we are addressing the costs of the parallelized block in Steps 1–6. In Steps 1 and 2, the data are distributed equally among all partitions, and within each partition, the Spark scheduler assigns available threads to the tasks required by Step 4. Let  $W_s$  denote the serial work

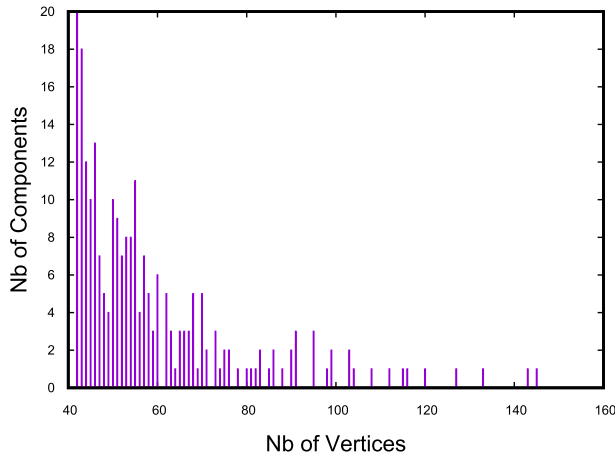


Fig. 2. Distribution of components with respect to the number of vertices (range from 40 to 140 vertices per component).

required by the algorithm in Steps 1–6. Because Step 4 engages all of the threads on the equally distributed data, its cost is accounted for by  $\Theta(W_s/p)$ . A parallel algorithm is work-optimal if it does not run more instructions than the sequential version, and as such, our algorithm is work-optimal. Step 6 is a communication step in which all threads in one partition communicate a pair of integers associated with each vertex they have been tasked with, and so, by the balanced data distribution of the earlier steps, the communication cost is  $\Theta(g \cdot |V|/p)$  flops. The only synchronization barriers needed are at the end of supersteps 1–4, after the communication step in 6. This concludes the proof.  $\square$

### E. Spatial Correlation Analysis

Here, we study the spatial correlation between the targeted nodes in the cascading attacks scenario in an attempt to uncover any underlying spatial pattern. For this purpose, and using the geocoding of the input vertices, we examine the subset of nodes  $F$  whose removal leads to a 90% connectivity loss. We are interested in measuring the relation between the failures separated by some given distance  $r$ . Following closely the spatial analysis of the 1996 blackout of Western Systems Coordinating Council area conducted in [19], we adopt the spatial correlation function  $C(r)$  that measures the relation between the failures that are  $r$  distance apart:

$$C(x) \propto \frac{\sum_{i,j \in F} (x_i - \bar{x})(x_j - \bar{x}) \delta(r_{ij} - r)}{\sum_{i,j \in F} \delta(r_{ij} - r)} \quad (4)$$

where  $x_i = 1$  if node is failed and 0 otherwise. Also,  $\bar{x}$  denotes the average number of cascading failures over the whole network,  $\delta$  denotes the function that singles out the  $j$  nodes that are  $r$  distance apart from  $i$ , and  $r_{ij}$  denotes the Euclidean distance between nodes  $i$  and  $j$ . Positive values for  $C(r)$  indicate a tendency of failures to be close to each other, while negative values indicate anticorrelations.

## IV. RESULTS

Our Spark program is written in Scala using Hadoop distribution 2.7.2 and run on a Intel Xeon machine with two

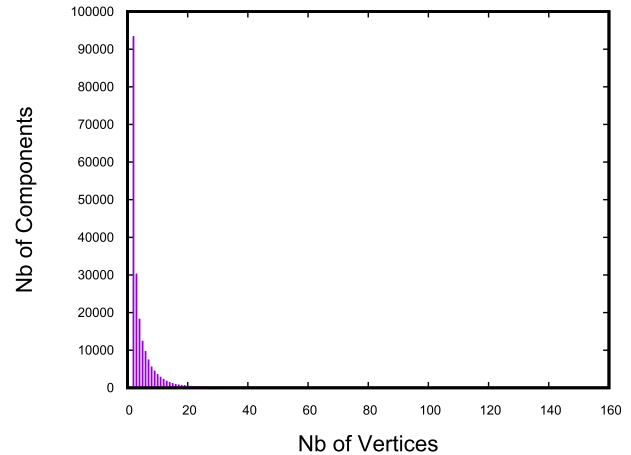


Fig. 3. Distribution of components with respect to the number of vertices.

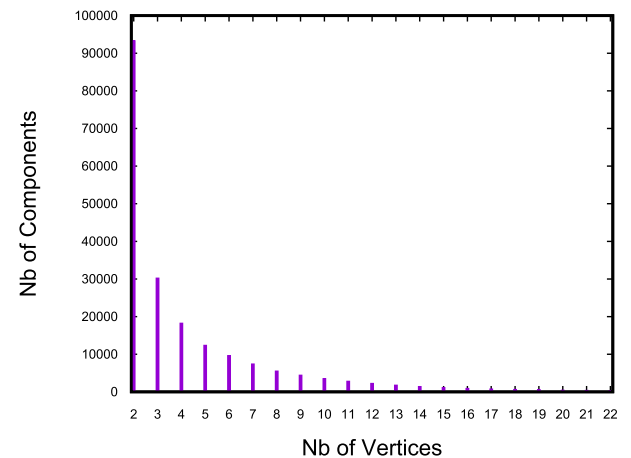


Fig. 4. Distribution of components with respect to the number of vertices (range from 2 to 20 vertices per component).

TABLE I  
PERCENTAGE OF VERTICES TO BE REMOVED TO REACH  
60% AND 80% LOSSES

Loss	BC	C	R	D
60%	11.6%	4.3%	59.8%	7.7%
80%	27.8%	14.6%	79.7%	29.5%

physical CPUs. Each CPU has 16 E5 – 2650 @ 2.00 GHz cores, giving us a total of 32 cores. The machine also has 32 GB of RAM and runs Ubuntu 16.04.

### A. Input Description

As discussed earlier, our input graph  $G$  is an undirected, unweighted graph with 679 965 edges and 899 162 vertices. Each vertex and edge is represented using 8 bytes, and therefore, the total amount of memory required by our graph is about 20 MB.

Our input graph also consists of 199 004 connected components, with the maximum component size at 145 and the minimum at 2. The distribution of component sizes is shown in Fig. 3, where the numbers of components with sizes above, say 40, were dwarfed by the components with smaller sizes. Fig. 2 (resp. Fig. 4) is an alternative figure that reveals the distribution of larger (resp. smaller) components.

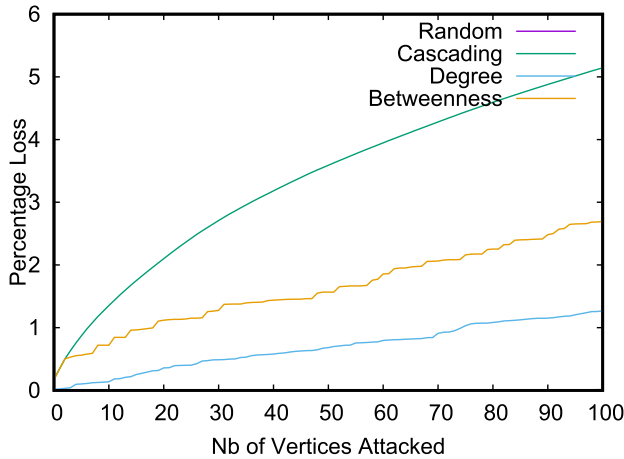


Fig. 5. Loss percentage: first 100 attacks.

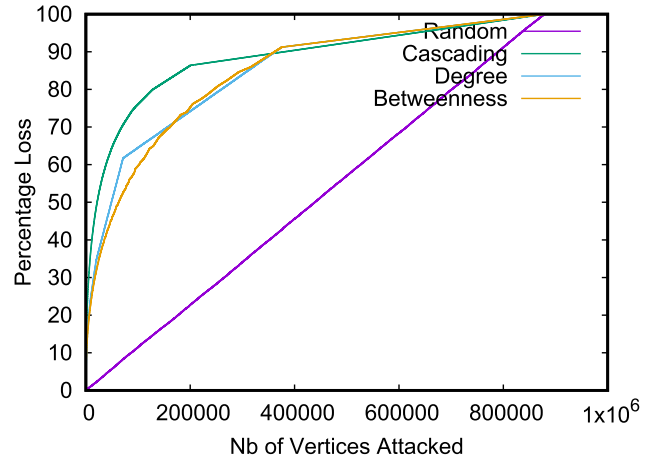


Fig. 7. Loss percentage: overall attacks.

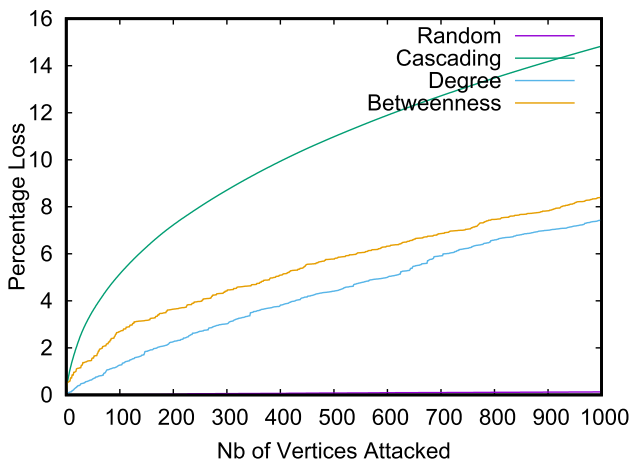
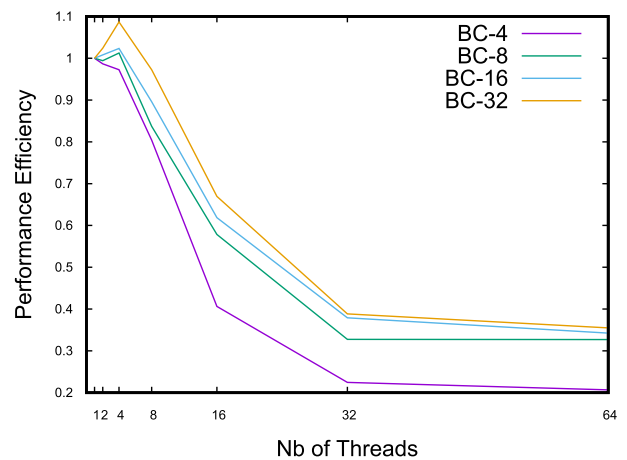


Fig. 6. Loss percentage: first 1000 attacks.

Fig. 8. Efficiency for the load-based (BC) scenario and graph  $G$ .

To investigate the scalability of our code for increasing input, we replicate the graph by doubling it (labeled graph  $G_2$ ) and then quadrupling it (labeled graph  $G_4$ ) while preserving the same structure as far as connected components are concerned.

### B. Failures and Connectivity Loss Analysis

For each of the four scenarios described in Section III, we vary the number of threads and partitions, compute the percentage of loss, and plot this value against the number of nodes removed from the graph. In the following tables and figures (Tables I–IV and Figs. 8–19),  $R$  corresponds to random failures,  $D$  to degree based failures, BC to betweenness centrality (load) failures, and  $C$  to cascading failures. Figs. 5 and 6 present a refined view demonstrating the progression of loss for the first 100 and 1000 failures on transmission and distribution nodes. Fig. 7 shows the loss for the entire failures. The connectivity loss is faint and proportional to the number of failures in the random case. It is worse for the cascading scenario, whereas the connectivity losses associated with the load-based versus degree-based scenarios are more or less comparable. These findings are confirmed in Table I, which presents, for each scenario, the percentage of failed nodes effecting in 60% and 80% (nearly total) connectivity loss. Some of these

figures, particularly, the relatively high percentages required in each of the BC,  $D$ , and  $R$  scenarios that precede total failure, can be interpreted to say that the Lebanese grid is highly redundant, and thus somehow resilient, thanks to its decomposition into numerous components and its reliance on local diesel generators that alleviate the effects of blackouts and failures.

### C. Run Time and Parallel Efficiency Analysis

The run time for each pair of ( $thread$ ,  $partition$ ) values using our input graphs is shown in Tables II–IV, for  $G$ ,  $G_2$ , and  $G_4$ , respectively. Moreover, the corresponding performance efficiency plots are shown in Figs. 8–19. Here, parallel efficiency is defined as  $(T_s/p \cdot T_p)$ , where  $T_s$  denotes the serial run time and  $T_p$  denotes the parallel run time given  $p$  parallel processes. The timings shown in Tables II–IV correspond to the parallel phase of the algorithm. The sequential run time was omitted, because it was extremely negligible compared with the parallel part that is of higher order (in contrast to linear running time in Steps 7 and 8). By default, Spark sets the partition size at 64 MB. This is too large for our given graph. As a result, we choose to override the default value by specifying the number of partitions at compile time. When this is done, Spark readjusts the size of each partition based

TABLE II  
RUN TIME (IN SECONDS) FOR GRAPH  $G$

Threads	BC-4	BC-8	BC-16	BC-32	C-4	C-8	C-16	C-32	D-4	D-8	D-16	D-32	R-4	R-8	R-16	R-32
64	27	17	22	41	53	34	32	52	26	15	20	41	23	17	20	39
32	24	17	19	37	52	33	31	49	20	16	18	36	20	19	19	36
16	24	18	22	39	53	35	37	54	21	16	19	37	19	16	20	37
8	25	25	30	54	55	50	56	80	21	21	26	51	20	20	26	50
4	41	41	53	97	87	86	98	138	33	33	45	89	30	31	44	90
2	82	84	108	206	165	165	189	287	65	66	92	188	55	62	87	186
1	161	167	219	423	312	320	369	543	124	127	187	384	104	114	174	344

TABLE III  
RUN TIME (IN SECONDS) FOR GRAPH  $G_2$

Threads	BC-4	BC-8	BC-16	BC-32	C-4	C-8	C-16	C-32	D-4	D-8	D-16	D-32	R-4	R-8	R-16	R-32
64	127	71	48	55	119	64	60	76	108	58	45	53	45	32	31	50
32	120	59	45	49	116	68	56	71	108	57	42	47	40	29	29	45
16	128	66	39	52	115	66	65	79	112	57	36	50	41	33	30	46
8	123	66	56	77	120	97	98	121	106	54	46	68	40	34	39	63
4	164	104	101	142	210	191	185	223	134	83	80	119	60	57	69	113
2	263	203	207	292	416	353	366	462	209	159	153	239	111	110	131	224
1	452	357	371	525	761	804	793	969	355	281	286	434	196	201	230	402

TABLE IV  
RUN TIME (IN SECONDS) FOR GRAPH  $G_4$

Threads	BC-4	BC-8	BC-16	BC-32	C-4	C-8	C-16	C-32	D-4	D-8	D-16	D-32	R-4	R-8	R-16	R-32
64	525	317	171	117	411	179	127	123	430	220	134	104	133	79	70	87
32	491	274	153	99	387	202	123	121	449	243	149	100	122	101	64	75
16	473	311	133	96	401	184	127	137	457	236	113	77	130	86	62	78
8	557	250	132	126	387	234	204	212	452	222	113	105	119	93	75	93
4	750	333	221	234	612	431	374	398	601	274	168	190	151	126	120	163
2	1041	526	430	467	1042	791	736	793	888	436	344	395	273	235	237	337
1	1626	902	879	1021	2156	1784	1577	1754	1584	905	730	783	533	458	456	647

on their total number as well as on the size of the input graph. The several connected components in each graph get mapped onto all partitions, such that the number of vertices in each partition is balanced across all partitions. We experiment with a number of partitions ranging from 4, 8, 16, and 32 in order to explore the effect that the number of partitions has on run time and parallel efficiency. From Figs. 3 and 2, we gather that there would be enough connected components in each partition to engage all threads assigned to the partition. Also, the fact that many connected components have sizes greater than, say, 40, ensures that the distributed work is more or less balanced.

As expected, for all three graphs, the fastest and yet the worst parallel efficiency correspond to the  $R$  (random failures) scenario that does not rely on any centrality measure computation. In contrast, the highest run times and, hence, best efficiency are for the  $C$  (cascading scenario) that updates the BC of all the nodes following each node removal. It is clear that the more work entailed by a certain scenario, the better it will be for the parallel program to compensate for the overheads associated with the partitioning and scheduling operations. Reading across the three tables for the same partition size and same scenario, our tool shows improved scalability as the input size grows.

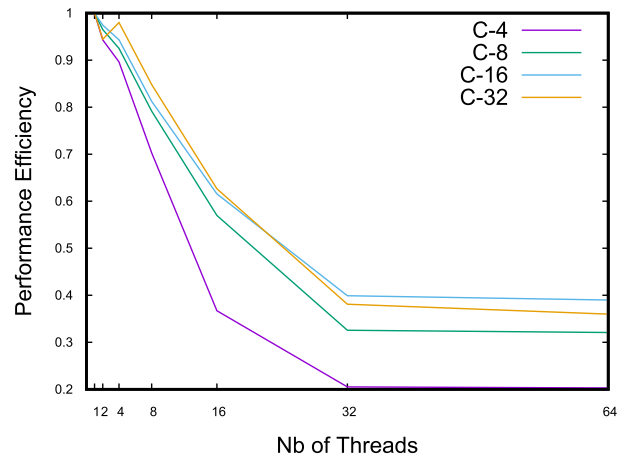
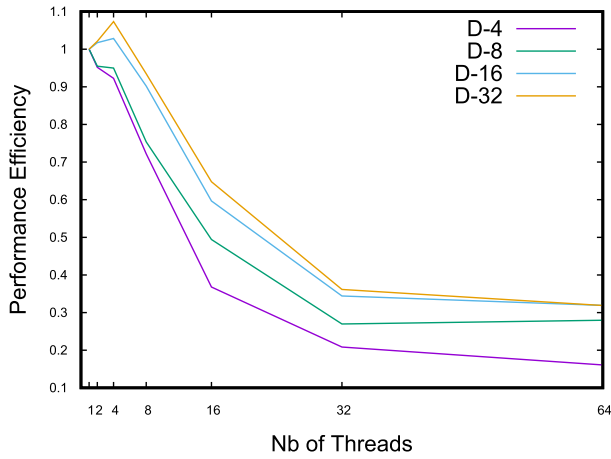
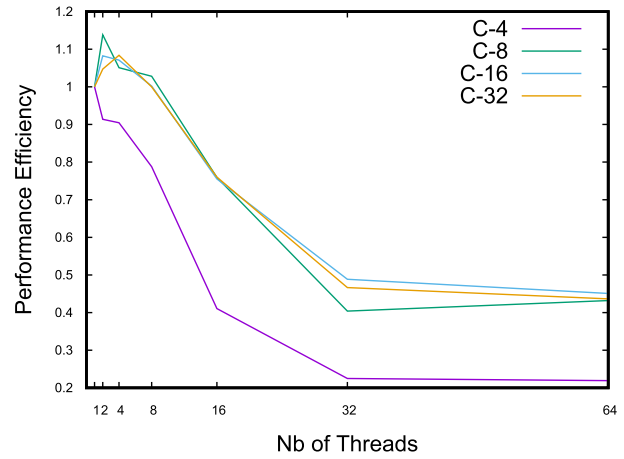
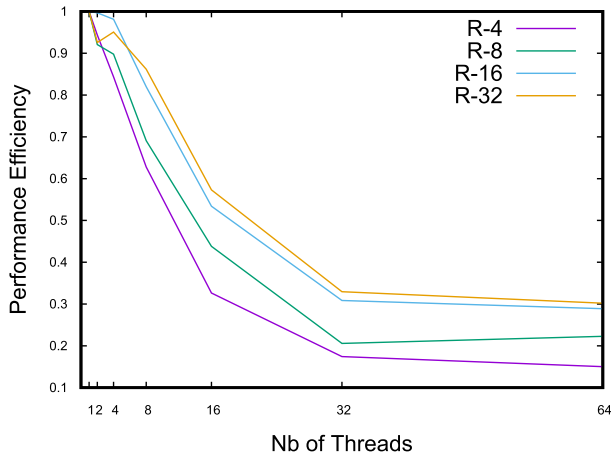
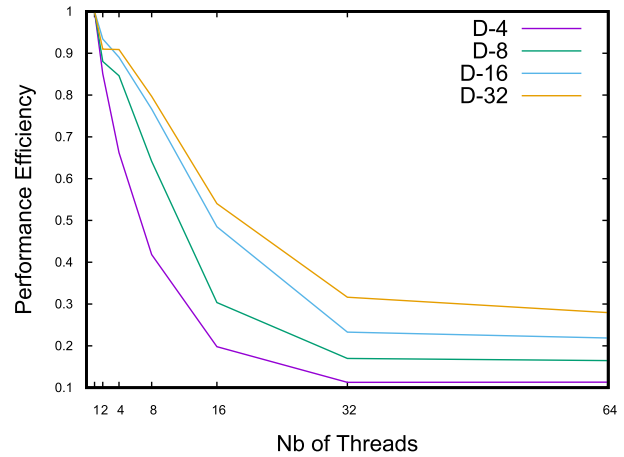
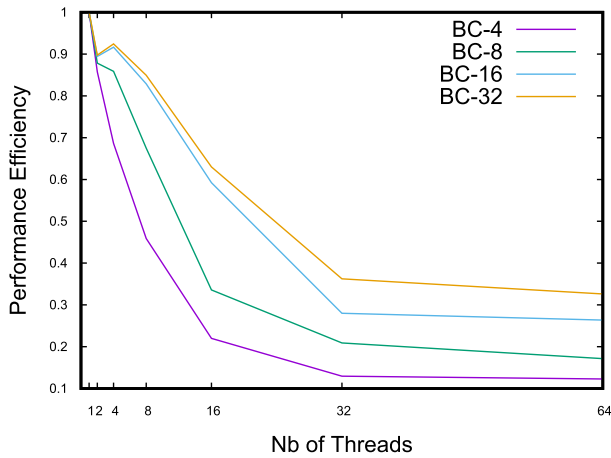
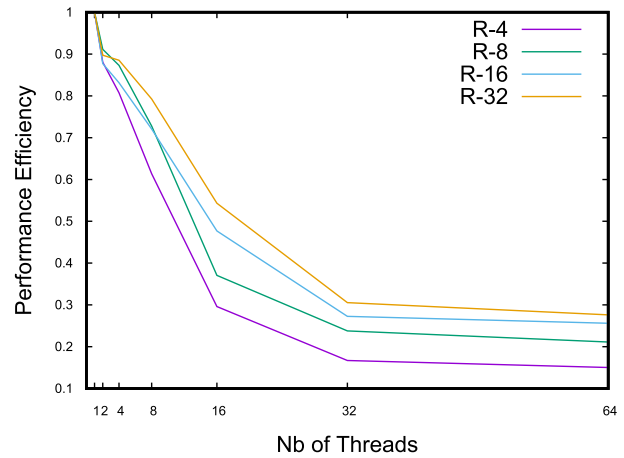


Fig. 9. Efficiency for the cascading (C) scenario and graph  $G$ .

For each given graph and each scenario employed, we notice that the actual run time has opposing trends that depend on the number of partitions. For Tables II–IV, there is a cutoff value for the number of partitions, before which run time continues to improve and after which it starts to deteriorate. For graph  $G$ , the cutoff number is at 8, for graph  $G_2$ , it is somewhere

Fig. 10. Efficiency for the degree-based ( $D$ ) scenario and graph  $G_1$ .Fig. 13. Efficiency for the cascading ( $C$ ) scenario and graph  $G_2$ .Fig. 11. Efficiency for the random ( $R$ ) scenario and graph  $G_1$ .Fig. 14. Efficiency for the degree-based ( $D$ ) scenario and graph  $G_2$ .Fig. 12. Efficiency for the load-based ( $BC$ ) scenario and graph  $G_2$ .Fig. 15. Efficiency for the random ( $R$ ) scenario and graph  $G_2$ .

between 8 and 16, and for  $G_4$ , it is at 16. We justify the improvement in run time as we move closer to the cutoff threshold as follows. With a higher number of partitions, one would expect that the multiple threads will be spread about, sharing the work but on data that are more split into distinct regions of main memory. As such, we have reduced contention over the shared address space, as well as scheduling costs associated with managing threads on one single partition.

We now argue that creating more partitions beyond the cutoff number drastically affects the performance and introduces a huge overhead—particularly when the number of threads is low. For instance, in Table II, if we consider only one thread of the  $BC$  scenario, it takes 161 (resp. 423) s in case of 4 (resp. 32) partitions. We attribute this overhead to the shuffling and repartition operations taking place at the end of each stage that assigns one or more threads from one partition to another.

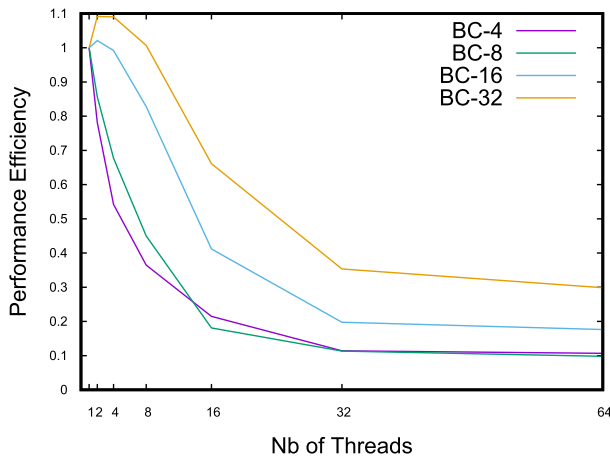


Fig. 16. Efficiency for the load-based (BC) scenario and graph  $G_4$ .

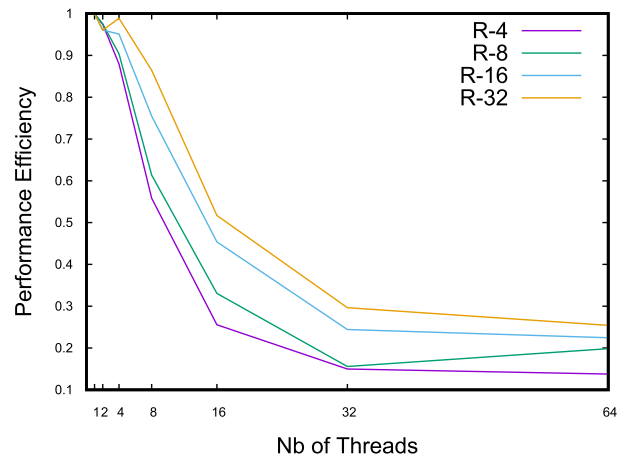


Fig. 19. Efficiency for the random (R) scenario and graph  $G_4$ .

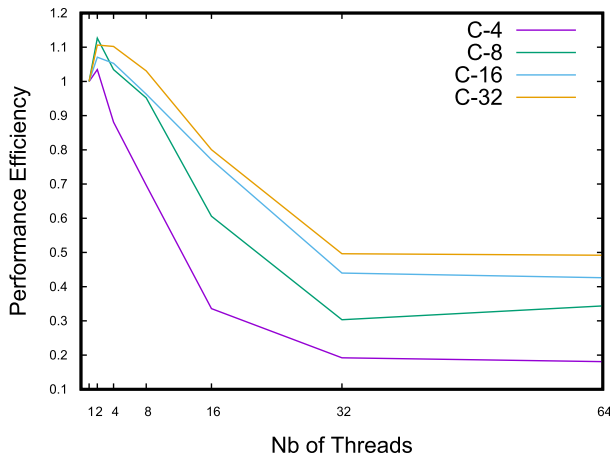


Fig. 17. Efficiency for the cascading (C) scenario and graph  $G_4$ .

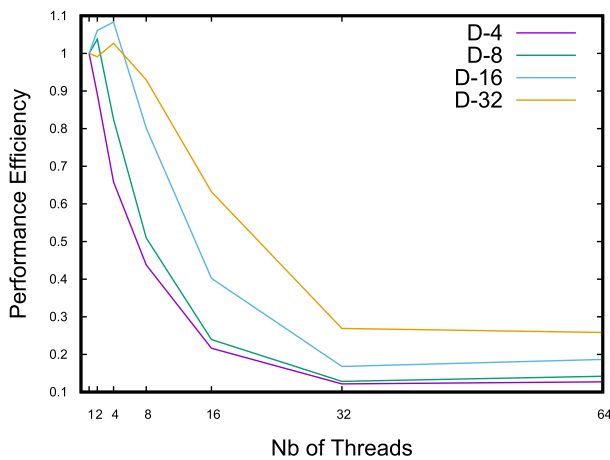


Fig. 18. Efficiency for the degree-based (D) scenario and graph  $G_4$ .

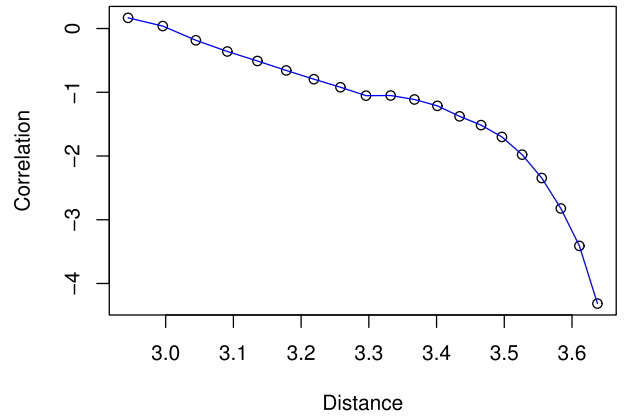


Fig. 20. Spatial correlation between the nodes in the cascading scenario on a logarithmic scale.

In that phase, some transformations (e.g., `groupByKey`, `reduceByKey`, and `join` operations) require shuffling and repartitioning of data, for instance, to group all the items with the same keys in the same partition.

With that said, we observe that efficiency actually improves for larger partitions. This is not to be construed, however, to mean that the parallelization has improved in any way, but rather that the rate of deterioration in the performance

for a smaller number of threads (particularly, the case of one thread) is higher when the number of partitions exceeds the cutoff number, resulting in a higher speedup as the number of threads grows larger.

Moreover, we notice that the parallel efficiency in some cases is above one (e.g., in the case of BC 32 partitions and 2 threads). This may be due to the effect of the garbage collectors. In the case of one thread and several partitions, the threshold of the garbage collector would be reached ahead of time, whereas in case of more threads/processes, less data need to be freed. Additionally, another cause behind superlinear speedup can be attributed to the “caching effect,” which results from the varying speeds in accessing different levels of the memory hierarchy on which the input graph and intermediate data are stored. As the number of threads increases, the data assigned to each thread decrease, rendering it into the smaller cache levels which are faster to access. As a result, the reduction in run time is not solely explained by the increase in the number of working threads but also in the reduction of the time spent on I/O, which causes the theoretical estimate for parallel speedup to go beyond  $p$  (given  $p$  threads), or equivalently, for parallel efficiency to go beyond 1.

#### D. Spatial Analysis

The spatial correlation analysis shown in Fig. 20 reveals a long-range correlated case with correlation decaying slowly with distance, particularly, in the linear regime, where  $C(r) \propto r^{-\gamma}$ ,  $\gamma = 1.13$ , which is in agreement with the literature results in [19]. Overload failures usually propagate through collective interactions among system components. Our results reveal that high failures in critical nodes have an impact that propagates across long path lengths on Lebanese soil.

#### V. CONCLUSION

A contingency analysis is a security function to assess the ability of a power grid to sustain various combinations of power grid component failures at energy control centers. To date, there exists no such work to examine the power grid resilience in Lebanon, a country which is still reeling under the effect of a brutal civil war and, recently, bearing an additional burden associated with the spillover from the Syrian war. This neighboring conflict has exposed Lebanon to a number of random terrorist attacks and caused it to become one of the major hosts of Syrian refugees, in addition to Iraqi and Palestinian refugees in former years. The strains on Lebanon's vital infrastructure and economical resilience have also been affecting the host community itself, where electric power supply is becoming increasingly drained. Our analysis of the resilience of the Lebanese power grid captures all of the networks down to its finest spatial coordinates. The computational burden associated with the resulting big graph is alleviated using a Spark-based, BSP modeled algorithm that balances the work optimally and incurs linear communication cost and a constant amount of synchronization among threads. Our analysis exploits a high level of decentralization in the Lebanese power grid and reveals a high level of redundancy, thanks to Lebanon's widespread reliance on diesel generators for surviving daily power blackouts. Our analysis also reveals the loss of connectivity in the power grid as a function of failures. This function behaves analogously to the case of the North American grid, as can be seen from [4], for example.

In addition to the functional requirements described in this paper, the nonfunctional requirements of our contingency analysis software provide for all of the following.

- 1) *Usability*: The targeted audience of the system includes governmental and other vital organizations. Our tool is intended to assist governments in mitigating any possible exploitation of the network and the networks that are incumbent on it, for example, the road network, commercial network, and telephony.
- 2) *Portability*: The system can run on any Hadoop configured machine, which can be a serial machine, a multi-core, or a cluster of machines.
- 3) *Availability*: The system is an open source project and can be downloaded from the developer's Github repository at <https://github.com/okm02/power-grid-analysis>.
- 4) *Capacity*: The capacity of the system adapts to the size of the graph. No further amendments are required,

provided that the input graph decomposes into a set of connected components each of which can be processed locally on a serial computer.

#### REFERENCES

- [1] M. E. J. Newman, "The structure and function of complex networks," *SIAM Rev.*, vol. 45, no. 2, pp. 167–256, 2003.
- [2] J. Gao, X. Liu, D. Li, and S. Havlin, "Recent progress on the resilience of complex networks," *Energies*, vol. 8, no. 10, pp. 12187–12210, Oct. 2015.
- [3] A. Bashan, Y. Berezin, S. V. Buldyrev, and S. Havlin, "The extreme vulnerability of interdependent spatially embedded networks," *Nature Phys.*, vol. 9, pp. 667–672, Aug. 2013.
- [4] R. Albert, H. Jeong, and A.-L. Barabási, "Error and attack tolerance of complex networks," *Nature*, vol. 406, no. 6, pp. 378–382, Jul. 2000.
- [5] E. Bompard, D. Wu, and F. Xue, "Structural vulnerability of power systems: A topological approach," *Electr. Power Syst. Res.*, vol. 81, no. 7, pp. 1334–1340, 2011.
- [6] L. Dueñas-Osorio and S. M. Vemuru, "Cascading failures in complex infrastructure systems," *Struct. Saf.*, vol. 31, no. 2, pp. 157–167, Mar. 2009.
- [7] D. Manik *et al.* (Sep. 2016). "Network susceptibilities: Theory and applications." [Online]. Available: <https://arxiv.org/abs/1609.04310>
- [8] R. Cohen, K. Erez, D. Ben-Avraham, and S. Havlin, "Breakdown of the Internet under intentional attack," *Phys. Rev. Lett.*, vol. 86, no. 16, pp. 3682–3685, Apr. 2001.
- [9] M. Rosas-Casals, S. Valverde, and R. V. Solé, "Topological vulnerability of the European power grid under errors and attacks," *Int. J. Bifurcation Chaos*, vol. 17, no. 7, pp. 2465–2475, 2007.
- [10] E. Bompard, R. Napoli, and F. Xue, "Analysis of structural vulnerabilities in power transmission grids," *Int. J. Critical Infrastruct. Protection*, vol. 2, nos. 1–2, pp. 5–12, 2009.
- [11] C. D. Brummitt, P. D. H. Hines, I. Dobson, C. Moore, and R. M. D'Souza, "Transdisciplinary electric power grid science," *Proc. Nat. Acad. Sci. USA*, vol. 110, no. 30, p. 12159, Jul. 2013.
- [12] L. Daqing, J. Yinan, K. Rui, and S. Havlin, "Spatial correlation analysis of cascading failures: Congestions and Blackouts," *Nature Sci. Rep.*, vol. 4, Jun. 2014, Art. no. 5381.
- [13] R. Albert, I. Albert, and G. L. Nekarado, "Structural vulnerability of the North American power grid," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 69, no. 2, pp. 025103-1–025103-4, Feb. 2004.
- [14] J.-W. Wang and L.-L. Rong, "Robustness of the western United States power grid under edge attack strategies due to cascading failures," *Saf. Sci.*, vol. 49, no. 6, pp. 807–812, 2011.
- [15] R. V. Solé, M. Rosas-Casals, B. Corominas-Murtra, and S. Valverde, "Robustness of the European power grids under intentional attack," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 77, no. 2, pp. 026102-1–026102-7, Feb. 2008.
- [16] G. C. Ejebe and B. F. Wollenberg, "Automatic contingency selection," *IEEE Trans. Power App. Syst.*, vol. PAS-98, no. 1, pp. 97–109, Jan. 1979.
- [17] A. O. Ekwue, "A review of automatic contingency selection algorithms for online security analysis," in *Proc. 3rd Int. Conf. Power Syst. Monitor. Control*, Jun. 1991, pp. 152–155.
- [18] S. Jin, Z. Huang, Y. Chen, D. G. Chavarría-Miranda, J. Feo, and P. C. Wong, "A novel application of parallel betweenness centrality to power grid contingency analysis," in *Proc. 24th IEEE Int. Symp. Parallel Distrib. Process. (IPDPS)*, Atlanta, GA, USA, Apr. 2010, pp. 1–7, doi: [10.1109/IPDPS.2010.5470400](https://doi.org/10.1109/IPDPS.2010.5470400).
- [19] L. Daqing, J. Yinan, K. Rui, and S. Havlin, "Spatial correlation analysis of cascading failures: Congestions and blackouts," *Sci. Rep.*, vol. 4, p. 5381, Jun. 2014. [Online]. Available: <https://www.nature.com/articles/srep05381>
- [20] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, 1990.
- [21] R. Cohen, K. Erez, D. Ben-Avraham, and S. Havlin, "Resilience of the Internet to Random breakdowns," *Phys. Rev. Lett.*, vol. 85, no. 21, p. 4626, 2000.
- [22] R. Cohen, K. Erez, D. Ben-Avraham, and S. Havlin, "Breakdown of the Internet under intentional attack," *Phys. Rev. Lett.*, vol. 86, no. 16, p. 3682, 2001.
- [23] D. S. Callaway, M. E. J. Newman, S. H. Strogatz, and D. J. Watts, "Network robustness and fragility: Percolation on random graphs," *Phys. Rev. Lett.*, vol. 85, no. 25, p. 5468, 2000.

- [24] P. Hines and S. Blumsack, "A centrality measure for electrical networks," in *Proc. 41st Annu. Hawaii Int. Conf. Syst. Sci.*, Jan. 2008, p. 185.
- [25] Z. Wang, A. Scaglione, and R. J. Thomas, "Electrical centrality measures for electric power grid vulnerability analysis," in *Proc. 49th IEEE Conf. Decis. Control (CDC)*, Atlanta, GA, USA, Dec. 2010, pp. 5792–5797, doi: [10.1109/CDC.2010.5717964](https://doi.org/10.1109/CDC.2010.5717964).
- [26] G. Bianconi and F. Radicchi, "Percolation in real multiplex networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 94, no. 6, pp. 060301-1–060301-5, Dec. 2016.
- [27] D. S. Callaway, M. E. J. Newman, S. H. Strogatz, and D. J. Watts, "Network robustness and fragility: Percolation on random graphs," *Phys. Rev. Lett.*, vol. 85, no. 25, p. 5468, 2000.
- [28] B. Karrer, M. E. J. Newman, and L. Zdeborová, "Percolation on sparse networks," *Phys. Rev. Lett.*, vol. 113, no. 20, p. 208702, Nov. 2014.
- [29] F. Radicchi, "Percolation in real interdependent networks," *Nature Phys.*, vol. 11, no. 7, pp. 597–602, Jun. 2015.
- [30] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proc. Nat. Acad. Sci. USA*, vol. 99, no. 12, pp. 7821–7826, Apr. 2002.
- [31] A. Cavagna *et al.*, "Scale-free correlations in starling flocks," *Proc. Nat. Acad. Sci. USA*, vol. 107, no. 26, pp. 11865–11870, 2010.
- [32] H. A. Makse, S. Havlin, and H. E. Stanley, "Modelling urban growth patterns," *Nature*, vol. 377, pp. 608–612, 1995.
- [33] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- [34] G. Malewicz *et al.*, "Pregel: A system for large-scale graph processing," in *Proc. SIGMOD*, 2010, pp. 135–146.
- [35] R. H. Bisseling, *Parallel Scientific Computation*. London, U.K.: Oxford Univ. Press, 2004.
- [36] J. M. D. Hill *et al.*, "BSPLib: The BSP programming library," *Parallel Comput.*, vol. 24, no. 14, pp. 1947–1980, 1998.
- [37] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph processing in a distributed dataflow framework," in *Proc. 11th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, Broomfield, CO, USA, Oct. 2014, pp. 599–613. [Online]. Available: <https://www.usenix.org/conference/osdi14/technical-sessions/presentation/gonzalez>
- [38] M. Bertolucci, A. Lulli, and L. Ricci, "Current flow betweenness centrality with apache spark," in *Proc. ICA3PP*, 2016, pp. 270–278.
- [39] H. Djidjev, S. Thulasidasan, G. Chapuis, R. Andonov, and D. Lavenier, "Efficient multi-GPU computation of all-pairs shortest paths," in *Proc. IPDPS*, May 2014, pp. 360–369.
- [40] N. Edmonds, T. Hoefler, and A. Lumsdaine, "A space-efficient parallel algorithm for computing betweenness centrality in distributed memory," in *Proc. HiPC*, Dec. 2010, pp. 1–10.
- [41] S. Jin, Z. Huang, Y. Chen, D. G. Chavarría-Miranda, J. Feo, and P. C. Wong, "A novel application of parallel betweenness centrality to power grid contingency analysis," in *Proc. IPDPS*, Apr. 2010, pp. 1–7.
- [42] A. G. Kumbhare, M. Frincu, C. S. Raghavendra, and V. K. Prasanna, "Efficient extraction of high centrality vertices in distributed graphs," in *Proc. HPEC*, Sep. 2014, pp. 1–7.
- [43] M. Redekopp, Y. Simmhan, and V. K. Prasanna, "Optimizations and analysis of BSP graph processing models on public clouds," in *Proc. IPDPS*, May 2013, pp. 203–214.
- [44] E. Solomonik, A. Buluç, and J. Demmel, "Minimizing communication in all-pairs shortest paths," in *Proc. IPDPS*, May 2013, pp. 548–559.
- [45] P. Boldi and S. Vigna, "Axioms for centrality," *Internet Math.*, vol. 10, nos. 3–4, pp. 222–262, 2014.
- [46] U. Brandes, "A faster algorithm for betweenness centrality," *J. Math. Sociol.*, vol. 25, no. 2, pp. 163–177, 2001.
- [47] L. E. Jordan and G. Alaghand, *Fundamentals of Parallel Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, 2002.
- [48] B. Parhami, *Introduction to Parallel Processing*. New York, NY, USA: Springer, 1999.

**Fatima K. Abu Salem** received the M.S. degree in mathematics from the American University of Beirut, Beirut, Lebanon, and the D.Phil. degree in computing from the University of Oxford, Oxford, U.K.

She is currently an Associate Professor with the Computer Science Department, American University of Beirut. Her current research interests include computer algebra, parallel computing, and data science for the public good.

**Mohamad Jaber** received the B.Sc. degree from Lebanese University, Beirut, Lebanon, in 2006, the M.Sc. degree from the University of Grenoble, Grenoble, France, in 2007, and the Ph.D. degree from the Verimag Laboratory, University of Grenoble, in 2010, all in computer science.

He is currently an Assistant Professor with the American University of Beirut, Beirut. His current research interests include distributed systems, software engineering, and big data analytics.

**Chadi Abdallah** received the Ph.D. degree in GIS and RS of natural resources from Pierre-and-Marie-Curie University, Paris, France, and the Professor degree from Lebanese University, Beirut, Lebanon, in 2016.

He has been a Researcher with the Lebanese National Council for Scientific Research (CNRS-L), Remote Sensing Center, since 1998. He was an Adjunct Researcher with the American University of Beirut, Beirut. He is currently an Associate Researcher and a Geologist with Pierre-and-Marie-Curie University and holds a post-doctoral position in radar interferometry at the University of Missouri, Columbia, MO, USA. He was a principle investigator on several projects related to water information management, modeling and decision-making; natural disasters; assessment, monitoring, and management of natural resources; and sustainable development. He was a Deputy Director and responsible of the Early Warning System Platform, CNRS-L. He has 14 years of teaching experience in hydrology, geology GIS, and remote sensing.

Dr. Abdallah was a member of the Disaster Risk Management Unit, Presidency of Councils. He was the Chair of the Arab Science and Technology Advisory Group for Disaster Risk Reduction and an Editorial Manager of the *Lebanese Science Journal*.

**Omar Mehio** received the B.Sc. degree in computer science from the American University of Beirut, Beirut, Lebanon, in 2017. He is currently pursuing the master's degree in computer science with the École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

His current research interests include big data analytics, deep learning, and optimization.

**Sara Najem** received the B.S. and M.S. degrees from the American University of Beirut, Beirut, Lebanon, with a focus on network complexity and dynamics, and the Ph.D. degree from McGill University, Montreal, QC, Canada, with a focus on out-of-equilibrium systems, all in physics.

She was a Post-Doctoral Fellow at the Couzin Lab, Princeton University, Princeton, NJ, USA, and a Gordon and Betty Moore Postdoctoral Fellow at GALCIT, California Institute of Technology, Pasadena, CA, USA. She is currently a Researcher with the National Center for Remote Sensing, CNRS-L, studying urban complexity with focus on energy and mobility.