Spatiotemporal contingency analysis for the Lebanese power grid using Apache Spark

Journal:	*IEEE Transactions on Computational Social Systems*
Manuscript ID	TCSS-2017-12-0106.R1
Manuscript Type:	Special Issue on Parallel and Distributed Processing for Computational Social Systems
Date Submitted by the Author:	28-Mar-2018
Complete List of Authors:	Abu Salem, Fatima; American University of Beirut Faculty of Arts and Sciences, Computer Science Jaber, Mohamad; American University of Beirut Faculty of Arts and Sciences, Computer Science Abdallah, Chadi; Conseil National de la Recherche Scientifique, National Center for Remote Sensing Mehio, Omar; American University of Beirut Faculty of Arts and Sciences, Computer Science Najem, Sara; Conseil National de la Recherche Scientifique, National Center for Remote Sensing
Keywords:	Spark, Power Grid, Graph, Spatiotemporal, Parallel and Distributed Computing

Answers to the Report for TCSS-2017-12-0106

Fatima K. Abu Salem[1], Mohamad Jaber[1], Chadi Abdallah[2], Omar Mehio[1], and Sara Najem[2]

[1] Computer Science Department, American University of Beirut, Beirut, Lebanon.
Email: {fa07, mj54, okm02}@aub.edu.lb
[2] National Center for Remote Sensing, National Council for Scientific Research (CNRS), Riad al Soloh, 1107 2260, Beirut, Lebanon.
Email: {snajem, chadi}@cnrs.edu.lb

Thank you very much for your letter regarding manuscript TCSS-2017-12-0106. Following the insights provided by the reviewers, we have taken into account all remarks and attempted to address all the questions. We hereby present a rebuttal (in italics) and attach a modified version of the paper. We have highlighted in red the modifications directly affecting the manuscript.

We wish to express our sincere gratitude to the reviewers and editors for helping us insignificantly improving the quality of our manuscript.

Sincerely Yours,

F. K. Abu Salem, M. Jaber, C. Abdallah, O. Mehio, S. Najem

1 Responses to the comments of Reviewer 1

- This paper proposed a vulnerability analysis of power grid based on graph topology and parallelized the analysis using Apache spark system. The problem is important since power grid is important to a country. The paper mentioned there are only a few works on such vulnerability analysis on power grid. This paper gives interesting exploration in this area. The drawback is that the approach is specific to the Lebanese power grid network. A more general approach will be more beneficial to the audience.

  The authors concede that a more general approach is certainly more beneficial; however, we still believe there is merit in tackling the specifics related to the Lebanese power grid, especially after Lebanon has withstood decades of wars and suffered from a lack of infrastructure rebuilding. With that said, our approach can generalize to networks of low connectivity such as that found in the Lebanese grid. We have clarified this in the Introduction.

- The paper removes the five graph nodes which are the five major power plants. This removal seems ad-hoc and is not formally included in their algorithm. It is better to give more graph property based quantatitive measure to remove nodes. This removal is very specific to the Lebanese network and it is not clear how to transfer it to other networks.

  Unfortunately, we received the data from the Lebanese ministry of power and energy in such a way that those five major power plants have already been removed. We just made this comment in Sec. III A, second paragraph. It is not clear to the authors whether we would still have to provide for how to remove those nodes, and we would appreciate it if the reviewer can let us know in case this is still required.

- The “parallel efficiency” is not defined.

  We now inserted a definition when this term is first used.

- It is better to give the runtimes for serial phase and parallel phase, since serial phase limits the maximum speedups when using more threads.

  Actually, the timings shown in the tables correspond to the parallel phase of the algorithm. The sequential part was omitted because it was extremely negligible compared to the parallel part that is of higher order (in contrast to linear running time in steps 7 and 8). We should have made this explicit and have by now inserted this comment in the text. Thank you for point it out.

- The parallelization proposed by the paper is a simple and direct parallelization approach. The parallelization effectiveness strongly depends on the number of connected components and the balance of the size of each component. In this paper, the authors observe each component is very small and the number of components are huge, which may hide the load balance problem. When the graph evolves or change to another network, it is not clear if this property still remains or not. I did not see how the partitioning of RDD is done in this paper(only different number of partitions is tried in the paper) and partitioning strategy may have strong impact on the parallelization.

  ln Spark, the number of partitions can either be (1) specified by the Spark execution engine so that to pick the best number of partitions with respect to the number of available cores and data size; or (2) can be adjusted (using `repartition` method) by the programmer according to the needs of the application. Nonetheless, in both cases, the partitions are created in a balanced way with respect to size. Moreover, GraphX uses a balanced graph partitioning to allow for the efficient computation by a BSP-like distributed computing framework. Additionnaly, Spark allows for the use of custom partitioning (using `HashPartitioner` and `RangePartitioner`), which can be defined by the programmer. In our case, we manually varied the number of partitions to study the scalability of our method, and for each partitioning scheme we made sure that the distribution of the partitions is done so that large strongly connected components are spread among different partitions. We added a paragraph (in Section III-D) in the paper to elaborate more about partitioning.

- The input graph in the paper is small and a single machine is enough for processing it. The spark system designed mainly for distributed environments brings extra cost when running on shared memory machines.

  We agree that the input graph is small, but what we are offering is a proof of concept that the algorithm scales and its performance improves should the graph get larger (for example, in studying the power grid of a bigger country, or networks of networks connected to the power grid such as telephony and banking system, to name a few examples). Also, whilst running a distributed algorithm written in Spark may induce significantly extra cost when run on shared memory, we believe this tends to be the case mainly when the algorithm introduces contention and cache issues such as false sharing on the shared data structures. This does not apply to our algorithm.

- In Table II-IV, no unit is specified for the runtime.

  We specified the unit, which is seconds, in Tables II to IV.

2 Responses to the comments of Reviewer 2

The following points specifically need to be addressed:

- The manuscript is not clear in defining the used terms and objectives:

- In abstract, it is not clear what the authors mean by “maintains optimal work”.

  Thank you for point this out. We have now clarified this in the proof of Prop. 5 as follows: A parallel algorithm is work optimal if it does not run more instructions than the sequential version.

- In Introduction, “to avoid data movement”.

  We have elaborated more on data movement, which implies network communications in case of distributed setting, garbage collector and memory allocation in case of local setting, as well as reads/writes across the memory hierarchy.

– In Section II, review of Resilience analysis using Graph Theory is missing some key aspects of analysis including percolation based methods and structural vulnerability analysis both respect to the connectivity properties and cascading failure studies including various propagation models on graphs.

We have updated Section II by discussing other aspects of analysis [1–4] such as percolation based method.

– The use of Spark is not well-motivated and adequately discussed in the Introduction. It is not clear what algorithm or analysis on graph the authors would like to apply to the power grid network that will require the use of Spark. Also, the Background Sections lacks literature review on power network analysis using Spark or MapReduce.

Indeed – the only algorithm that requires the use of Spark is connected components. By large, this isn't a task parallel algorithm. Spark is used to distribute the data and so in a way it implements a data parallel rather than task parallel algorithm. Our objective has been to use serial graph algorithms on distributed data in SPMD mode (single program, multiple data). Please note that we have now inserted these remarks in the introduction. We have also noted that no other work uses spark or MapReduce for power grid analysis beyond the ones mentioned in the article (for example, Jin et als, 2010).

– The cascade model is not clear; is it load based or probabilistic or...? In page 4, one of the mechanism to select victim node is defined to be Cascade Attack. What does that mean? Cascade can be triggered but the process of cascade happens due to interactions among components (e.g., load redistribution) not by external force.

The cascading scenario is based on 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. We have updated the text (in Section II-A) to elaborate more on this.

– The physics of the power and other properties of the power system are not mentioned, discussed or considered in the study. A large body of work is emerging on advantages and limitations of pure graph-based analysis for power systems, which need to be mentioned.

We have added (in Section II-A) extra references [5, 6] that discuss the physical properties of the network.

– Editorial Comments:
  - Please don't use abbreviations without first defining them. Example: BSP in the abstract. Also, add a reference for that.

We revisited the paper by defining abbreviation first. Also, we added a reference for BSP in the introduction.

- For attackGraph and all the codes presented in the manuscript, it's better to include the pseudocode in the algorithm format. The current format is not standard!

  We re-wrote all the algorithms by following a standard formatting.

References

1. G. Bianconi and F. Radicchi, "Percolation in real multiplex networks," *Physical Review E*, vol. 94, no. 6, pp. 060 301–5, Dec. 2016.
2. D. S. Callaway, M. Newman, S. H. Strogatz, D. W. P. r. letters, and 2000, "Network robustness and fragility: Percolation on random graphs," *APS*.
3. B. Karrer, M. E. J. Newman, and L. Zdeborová, "Percolation on Sparse Networks," *Physical Review Letters*, vol. 113, no. 20, pp. 211–5, Nov. 2014.
4. F. Radicchi, "Percolation in real interdependent networks," *Nature Physics*, vol. 11, no. 7, pp. 597–602, Jun. 2015.
5. P. Hines and S. Blumsack, "A Centrality Measure for Electrical Networks," in *2008 The 41st Annual Hawaii International Conference on System Sciences*. IEEE, Sep. 2007, pp. 185–185.
6. Z. Wang, A. Scaglione, and R. J. Thomas, "Electrical centrality measures for electric power grid vulnerability analysis," in *Proceedings of the 49th IEEE Conference on Decision and Control, CDC 2010, December 15-17, 2010, Atlanta, Georgia, USA*, 2010, pp. 5792–5797. [Online]. Available: https://doi.org/10.1109/CDC.2010.5717964

Spatiotemporal contingency analysis for the Lebanese power grid using Apache Spark

Fatima K. Abu Salem[2], Mohamad Jaber[2], Chadi Abdallah[1], Omar Mehio[2], and Sara Najem[1]

***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 decentralisation via numerous connected components. Our Apache Spark implementation simulates 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 (BSP) model and maintains optimal work, linear communication time, and a constant number of synchronisation barriers. We complement our work with a spatial understanding of the exposed hotspots. Our results reveal that failures in the power grid are spatially long-range correlated, and that correlations decay with distance. In a couple of attack scenarios our Spark implementation achieves significant speedup on 16 cores for a graph with about 9×10^5 nodes. Scalability towards 32 nodes improves when experimenting with replicas of the power grid graph that are double and quadruple the original size. This renders our work suitable to larger networks at many vital levels beyond the power grid.**

***Index Terms*—Spatiotemporal, Spark, Power Grid, Parallel Computing, Graph**

I. Introduction

In scale-free networks (SF) 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], [2], [3], [4]. Examples of this class of SF are the Internet, power systems, and transportations 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], [6], [7], [8].

In power systems, and unlike random failures which emerge locally, blackouts are severe events associated with cascading behavior leading to global network collapse [9], [10], [11], [12], [13], [14], [15]. Examples such as the one affecting the north-east in the US and eastern Canada in 2003 burdened the respective economies with ten billion dollars of direct costs [12]. Such failures can be linked to either structural dependencies, 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 (NERC) [18], and obviously do not capture the overload failures that propagate through interactions among a system's physical components. It thus becomes necessary to be able to perform 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], [19], the power grid, particularly the network of its transmission lines, is analyzed using graph algorithms such as betweenness centrality 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 towards precision-based policy making and disaster response in a region marred by wars and under-development.

Using elements from distributed algorithm design as well as spatial analysis, our work extends the approach of [4] in analysing 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 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). The choice of Spark also renders our system to be suitable for 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) model [20], [36], [37], [38] and employs serial graph algorithms on distributed data in SPMD mode (single program, multiple data). Our data parallel algorithm maintains parallel optimal-work, linear communication time, and a constant

[1] National Center for Remote Sensing, National Council for Scientific Research (CNRS), Riad al Soloh, 1107 2260, Beirut, Lebanon. Email: {snajem, chadi}@cnrs.edu.lb.

[2] Computer Science Department, American University of Beirut, Beirut, Lebanon. Email: {fa07, mj54, okm02}@aub.edu.lb.

Equally contributing author. Corresponding author: Sara Najem.

number of synchronisation barriers.

Our manuscript is organized as follows. In Sec. 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 we describe some intrinsics related to Apache spark for big data distributed processing. In Sec. 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 Betweenness Centrality, orchestrated using four scenarios known in the literature [4]. Each of these scenarios simulates a unique temporal mode of removal of vertices. Despite that our approach does not generalise to arbitrary networks, we believe there is merit in tackling the specifics related to the Lebanese power grid, especially after Lebanon has withstood 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 Sec. IV, we perform 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 majority of connected components have a relatively small cardinality. In fact, scalability towards 32 nodes improves when experimenting with replicas of the power grid graph that 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 decentralised 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 that correlations decay with distance. In Sec. V, we conclude with remarks around the impact of our work.

II. Background

A. Resilience analysis using Graph Theory

The seminal works in [4], [19] address contingency analysis for the power grid using simulation of complex networks. Typically, the power grid is modeled using generator, transmission, and distribution nodes. In such scale-free networks, a large fraction of the nodes have low degrees and a substantially smaller fraction exhibit high degrees. Those networks are established to be resilient against random failures upon nodes, and the overall loss to be contingent upon the targeting of the high-degree hubs [21], [22], [23], [24]. In addition to 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 [25], [26].

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 [27], [28], [29], [30]. 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 by [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 betweenness centrality. A node's betweenness centrality 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 minimising the number of edges that the path passes through. Else, the path is obtained by minimising the sum of the weights of the edges that the path passes through. Given an arbitrary vertex in a graph, its betweenness centrality is defined to be the number of shortest paths that pass through the given vertex. Exploiting betweenness centrality 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 re-examined 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 betweennes centrality and (4) decreasing order of node betweenness centrality in cascading order resulting from the nodes' dynamical removal. The cascading scenario is based on recalculated centrality measures made available after the identification of the most central node in the initial graph. Subsequent to its removal a new graph ensues. The central node of the new graph should then be identified and the process iterates.

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 there is based on applying edge betweenness centrality [31] to the power grid topology. High-impact components in the power grid are defined using the most traversed edges. We believe this approach is less exhaustive than the one adopted by [4], and we 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 [32], [33]. As a leading example that we follow, this measure is used in [19] to measure the relation between failures separated at some distance r. More details follow in Sec. III.

B. Distributed Computation frameworks

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

GraphX.

1) MapReduce and Pregel: The last few years have witnessed an uptake in distributed data processing research. Among the leading frameworks to exploit distributed computation on commodity hardware is the MapReduce paradigm [34]. 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 serialisation in the parallel code.

The first serious development for supporting graph algorithms using the MapReduce framework is found in Google's Pregel [35]. Instead of coordinating the steps of a chained MapReduce program, Pregel is able to process iteration over supersteps under the BSP. 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 values stored in the local, fast memory of the processor.
2) A communication superstep: the processes exchange data between themselves if needed for the aggregation of the results computed in (1) above.
3) A barrier synchronisation superstep: each processor halts until all other processes have reached the same barrier.

According to this model, a graph algorithm in Pregel is organised as a sequence of iterations, and can be described from the point of view of a vertex, that manages its state and sends messages only to its neighbours. 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 thanks to 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 RDDs (resilient distributed datasets), which can be distributed across a cluster. These RDDs are immutable, partitioned data structures that can be manipulated through multiple operators like Map, Reduce, and Join. For example, RDDs are created through parallel transformations (e.g., map, group by, filter, join, 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 a 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 analyse 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 [39] 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 dataset contains 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.

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 power and energy 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. Our analysis simulates attacks on both transmission as well as distribution substations. It also pursues inter-dependencies as

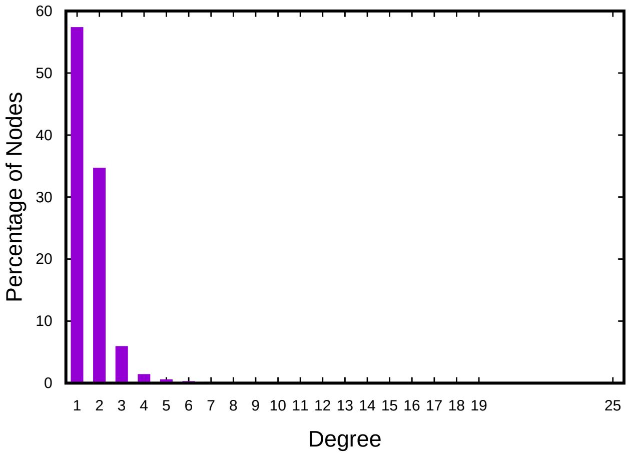

Fig. 1. Degree of vertices

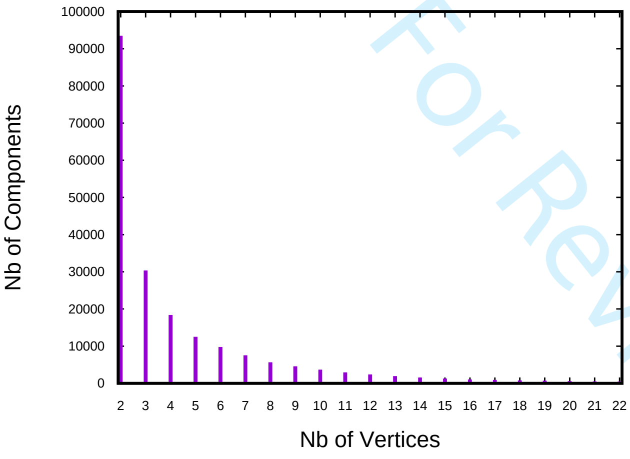

Fig. 2. Distribution of components w.r.t. the number vertices (range from 2 to 20 vertices per component)

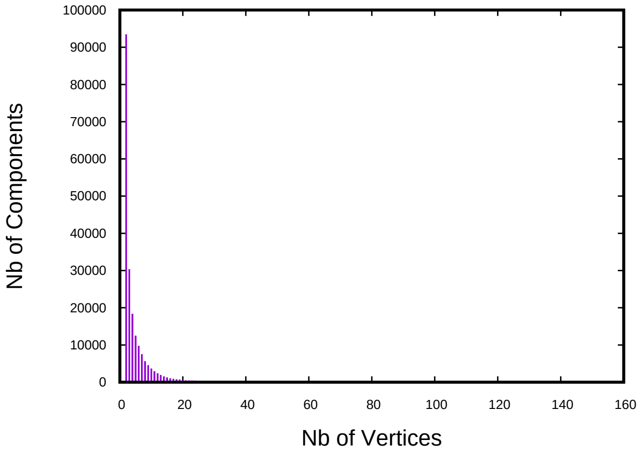

Fig. 3. Distribution of components w.r.t. the number vertices

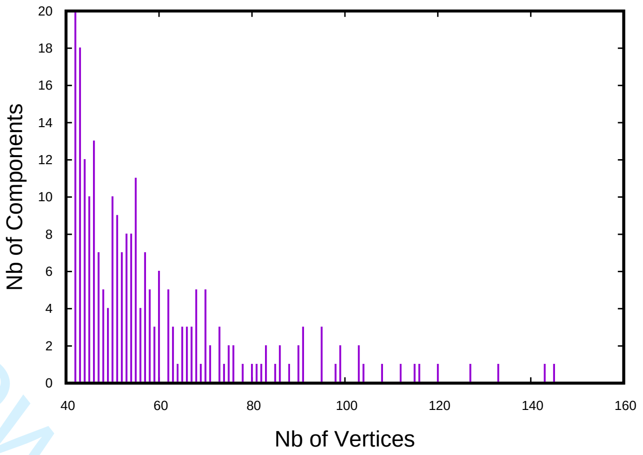

Fig. 4. Distribution of components w.r.t. the number vertices (range from 40 to 140 vertices per component)

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 idealised (and simplified) view that capacity across the transmission lines is never jeopardised, and thus failure of the network is a result of attacks on substations only.

Our resulting network model consists of an undirected, unweighted graph with 679965 edges, representing transmission/distribution lines, and 899162 vertices, representing transmission/distribution substations. This is one order of magnitude larger than the graph treated in [4], and is 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 (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 notions of node attack, followed by an evaluation of the loss in power flow capacity across the entire network.

Algorithm 1: Random and cascading attacks

```
Input: Graph graph = (V, E)
Output: random and cascading attacks of the input graph
Function attackGraph(graph):
    for $i \leftarrow 0$ to $|V|$ do
        select victim vertex v;
        remove vertex v;
        update loss with respect to v;
    end
End Function
```

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 [40] for a comprehensive survey. For undirected graphs, the geometric measure related to the indegree of a given node, as well as the path-based measure based on its betweenness, have both yielded a contextual understanding of connectivity loss in power grids ([4], [18]). In line with this body of work, we select the victim vertex according to one of the following four scenarios:

- *Random attacks*: a vertex is selected at random.
- *Attacks based on geometric centrality measure*: the vertex with the highest degree is selected.
- *Attacks based on path-based centrality measure*: the vertex with the highest betweenness centrality is selected. The betweenness centrality of the nodes is only computed once (i.e., is not be updated after removing a vertex).
- *Cascading attacks*: This is similar to the betweenness centrality scenario; however, the betweenness centrality of the nodes is updated at each iteration (i.e., after removal of a victim vertex).

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

Algorithm 2: Scenario-based victim selection

```
Input: Graph graph = (V, E), boolean cascading
Compute betweenness centrality for each v in V;
Function attackGraph(graph, cascading):
    for $i \leftarrow 0$ to $|V|$ do
        select victim vertex $v$ in $V$;
        remove vertex $v$;
        if cascading then
            for each node $u$ not removed so far do
                recompute betweenness centrality for $u$;
            end
        end
        update loss with respect to $v$;
    end
End Function
```

The choice behind node betweenness centrality is motivated by the following. Let $\sigma(s,t)$ denote the number of shortest paths between two vertices s and t, and let $\sigma(s,t\,|\,v)$ denote the number of shortest paths between s and t that pass through v. A most recent betweenness centrality (BC) score of v is suggested by Brandes in [41] as follows:

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

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 that 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 which improves on former cubic running time algorithms. The algorithm follows an accumulation technique that invokes a single-source shortest paths (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 breadth first traversals. To illustrate, let

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

denote the dependency 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 dependency of s on v, which is the sum of dependencies 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 [41]. 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 localised 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 render 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 decentralised version of the grid as such. This distribution of the network leads us to readdress the connectivity/loss and betweenness centrality in a manner that explores the effects of local changes on the global graph properties. We address those two issues separately below.

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 so their location may remain “unknown” to an attacker, and 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:

$$\texttt{connectivity}(G) = \sum_{v \in V} \mid \texttt{reachable}(v) \mid,$$

where $\texttt{reachable}(v)$ 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:

$$\begin{aligned} &\texttt{connectivity}(G) \\ &= \mid V \mid \times \mid V - 1 \mid && \text{if } G \text{ is connected,} \\ &= \textstyle\sum_{i \in \{1,\ldots,n\}} \texttt{connectivity}(G_i) && \text{otherwise.} \end{aligned} \qquad (2)$$

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

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

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 $\texttt{loss}(G, x) = \texttt{connectivity}(G) - \texttt{connectivity}(G \setminus \{x\})$. The following proposition reveals how the local loss within a component translates to global loss:

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

$$\begin{aligned} &\texttt{connectivity}(G) - \texttt{connectivity}(G \setminus \{x\}) \\ &= \texttt{connectivity}(G_i) - \texttt{connectivity}(G_i \setminus \{x\}) \end{aligned} \qquad (3)$$

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

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

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

We now use this equation in:

$$\begin{aligned} &\texttt{connectivity}(G) - \texttt{connectivity}(G \setminus \{x\}) \\ &= \textstyle\sum_{j=1}^{n} \texttt{connectivity}(G_j) \\ &- \Big(\texttt{connectivity}(G_i \setminus \{x\}) + \textstyle\sum_{j=1, j \neq i}^{n} \texttt{connectivity}(G_j)\Big) \\ &= \texttt{connectivity}(G_i) - \texttt{connectivity}(G_i \setminus \{x\}) \end{aligned}$$

□

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 $\texttt{CC}(G) = \{G_1, \ldots, G_n\}$ denote the connected components of the undirected power grid graph G. Let $\deg_G(v)$ denote the degree of a node v in G, and $\deg_{G_i}(v)$ denote its degree in G_i, for some connected component $G_i \in \texttt{CC}(G)$ containing v. We then have:*

$$\deg_G(v) = \deg_{G_i}(v).$$

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 betweenness centrality measure:

Proposition 3. *Let $\texttt{CC}(G) = \{G_1, \ldots, G_n\}$ 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 $G_i \in \texttt{CC}(G)$ containing v. We then have:*

$$\texttt{C}_G(v) = \texttt{C}_{G_i}(v).$$

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 $\sigma(s, t \mid v)$, 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, $\sigma(s, t \mid v) = 0$. Using Eq. 1 for the BC score, we obtain that $\texttt{C}_G(v) = \texttt{C}_{G_i}(v)$ in all cases. □

D. Spark-based Implementation

We now present our Spark-based algorithm, guided by the propositions above. The many connected components of the power grid provide for an inherent data distribution of the graph which, as we will see in Sec. IV, allows for a balanced workload as well as locality of reference. Spark employs a storage abstraction called Resilient Distributed Datasets (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 a RDD. The execution engine can co-schedule 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 strongly connected component distributed kernel. This creates an RDD, `rddCC`, of items, where each item corresponds to a connected component.
3) Partition `rddCC` into several p partitions, for $p = 4, \ldots, 32$, $p \leftarrow 2 \times p$. 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 (ρ, λ) is appended to the end of a list $\mathbf{L}_t$. Here, ρ denotes its rank in the removal process if the vertex has been chosen in random order. Otherwise, ρ denotes its centrality (degree or load based). Also, λ 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 $loss(G, x_i) = \sum_{j=1}^{i} \lambda_i$. This step is also performed serially.

We now have the following:

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

Proof. We first address Step 4. By Prop. 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 Prop. 3, updating the betweenness centrality 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 parallelise the computation of all betweeness 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 Prop. 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 $\mathbf{L} = \{x_1, \ldots, x_n\}$, where $n = |L|$. 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 $loss(G \setminus \{x, y\})$ denote the loss in the graph after removing x followed by y. The losses returned in Step 7 are thus captured by $loss(G \setminus \{x_1, \ldots, x_n\})$:

- 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, $loss(G \setminus \{x, y\}) = loss(G \setminus \{x\}) + loss(G \setminus \{y\})$, i.e. the reduction operator on the individual losses can take place after the computation of individual losses locally in Step 4.
- 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 $loss(G \setminus \{x, y\})$ is computed correctly using the formula $loss(G \setminus \{x\}) + loss\left((G \setminus \{x\}) \setminus \{y\}\right)$, which has already been computed locally in Step 4.

□

Proposition 5. *The parallelised block in Steps 1–6 of the above algorithm maintains optimal parallel work, and requires linear time communication costs and a constant number of synchronisation 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 ℓ denote the machine-dependent cost in flops for all threads to synchronise. We then have:*

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

Proof. Here, we are addressing the costs of the parallelised block in Steps 1–6. In Steps 1 and 2, the data is 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 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 $\Theta\left(\frac{W_s}{p}\right)$. 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 $\Theta\left(g \cdot \frac{|V|}{p}\right)$ flops. The only synchronisation barriers needed are at the end of supersteps 1–4, and after the communication step in 6. This concludes the proof. □

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 lead to a 90% connectivity loss. We are interested in measuring the the relation between 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 failures that are r distance apart:

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

Here, $x_i = 1$ if node is is failed, and 0 otherwise. Also, $\bar{x}$ denotes the average number of cascading failures over the whole network, δ 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 anti-correlations.

IV. Results

Our Spark program is written in Scala using Hadoop distribution 2.7.2, and run on a Intel (R) Xeon machine with two physical CPUs. Each CPU has 16 $E5-2650$ @ 2.00GHz cores, giving us a total 32 cores. The machine also has 32GB of RAM, and runs Ubuntu 16.04.

A. Input description

As discussed earlier, our input graph G is an undirected, unweighted graph with 679965 edges and 899162 vertices. Each vertex and edge is represented using 8 bytes and so the total amount of memory required by our graph is about 20MB.

Our input graph also consists of 199004 connected components, with the maximum component size at 145 and the minimum at 2. The distribution of component sizes is shown in Fig. 8, where the numbers of components with sizes above, say 40, were dwarfed by the components with smaller sizes. Fig. 9 is an alternative figure that reveals the distribution of larger components.

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

B. Failures and connectivity loss analysis

For each of the four scenarios described in Sec. 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 all the tables and figures below, R corresponds to random failures, D to degree based failures, BC to betweenness centrality (load) failures, and C to cascading failures. Fig. 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.

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%

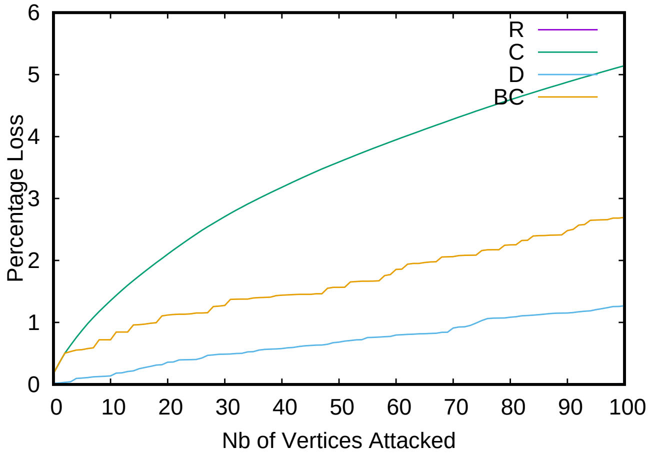

Fig. 5. Loss percentage: first 100 attacks

C. Run-time and parallel efficiency analysis

The run-time and parallel efficiency for each pair of (*thread,partition*) values using our input graphs are shown in

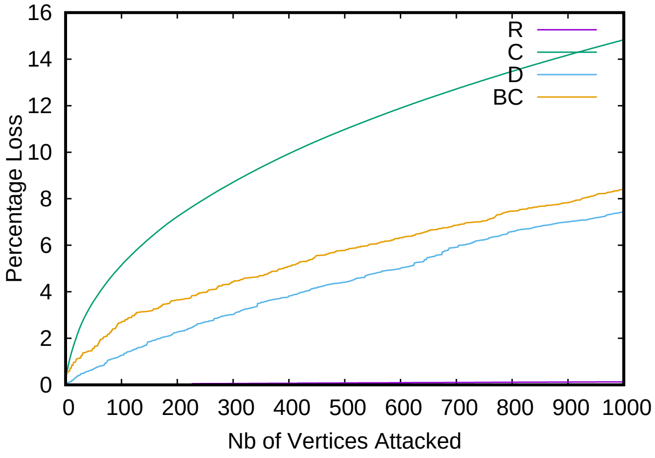

Fig. 6. Loss percentage: first 1000 attacks

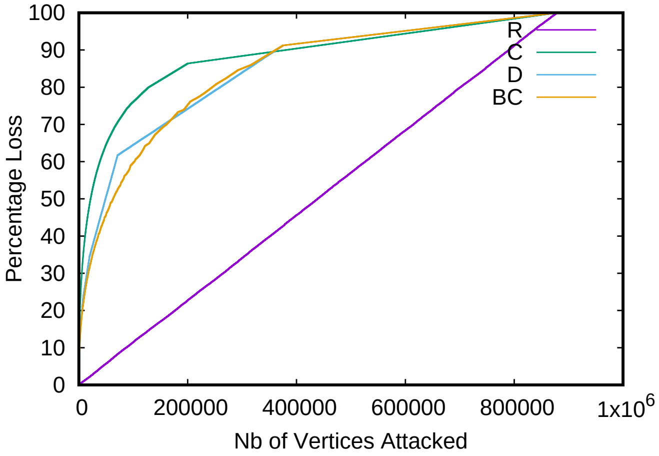

Fig. 7. Loss percentage: overall attacks

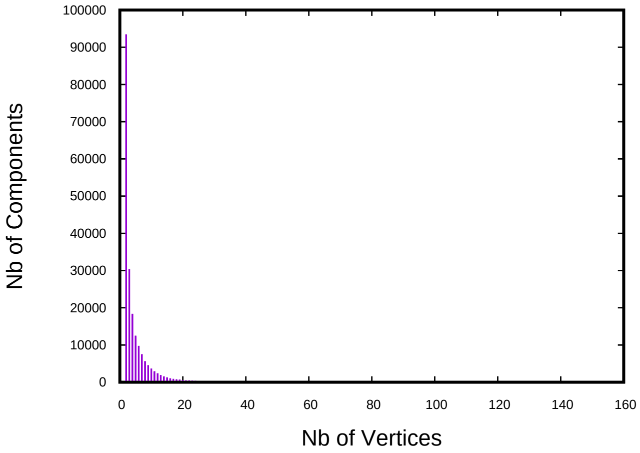

Fig. 8. Components size – overall distribution

Table II, III, and IV, for G, G_2, and G_4 respectively. Here, parallel efficiency is defined as $\frac{T_s}{p \cdot T_p}$, where T_s denotes the serial run-time and T_p denotes the parallel run-time given p parallel processes. The timings shown in the Tables correspond to the parallel phase of the algorithm. The sequential run-time was omitted because it was extremely negligible compared to 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 re-adjusts the size of each partition based 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 have on run-time and parallel efficiency. From Fig. 8 and 9, 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 betweenness centrality 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.

For each given graph and for each scenario employed, we notice that the actual run-time has opposing trends that depend on the number of partitions. For all three tables, there is a cut-off value for the number of partitions, before which run-time continues to improve, and after which it starts to deteriorate. For graph G, the cut-off number is at 8, for graph G_2 it is somewhere 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 cut-off 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 is 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 cut-off 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) seconds 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. In that phase, some transformations (e.g., `groupByKey`, `reduceByKey`, `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 isn't to be construed, however, to mean that the parallelisation 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 cut-off number, resulting in a higher-speedup as the number of threads grows larger.

D. Spatial analysis

The spatial correlation analysis shown in Fig. 22 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].

V. Conclusion

Contingency analysis is a security function to assess the ability of a power grid to sustain various combinations of

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

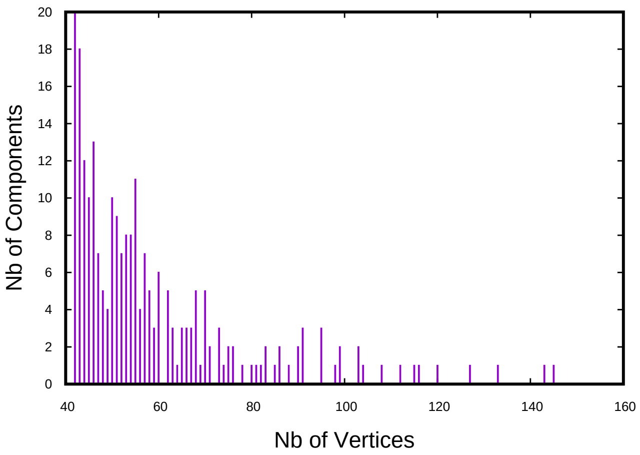

Fig. 9. Components size – cropped distribution

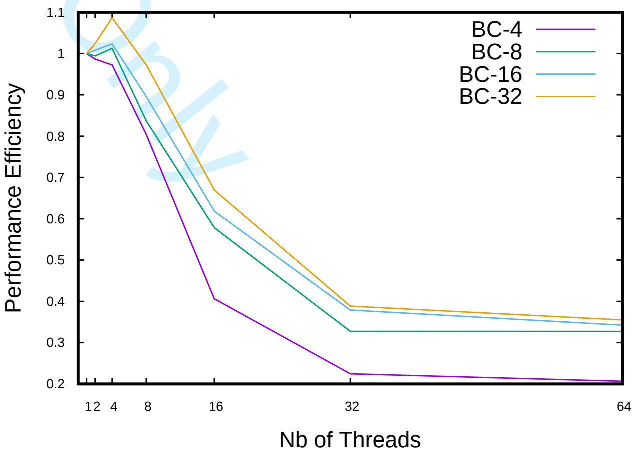

Fig. 10. Efficiency for the load based (BC) scenario and graph G

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 neighbouring 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 Lebanese power grid captures all of the network down to its finest spatial coordinates. The

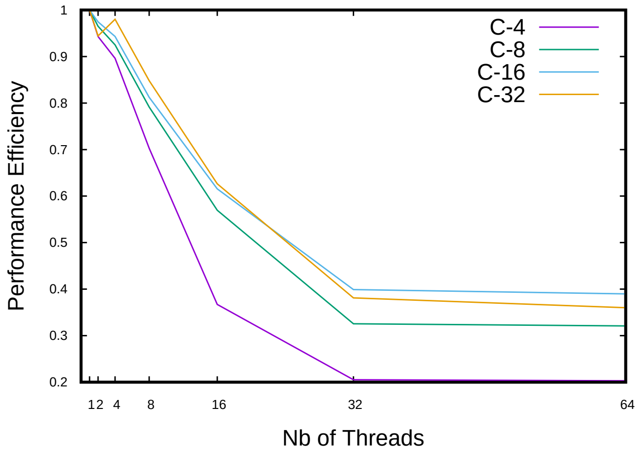

Fig. 11. Efficiency for the cascading (C) scenario and graph G

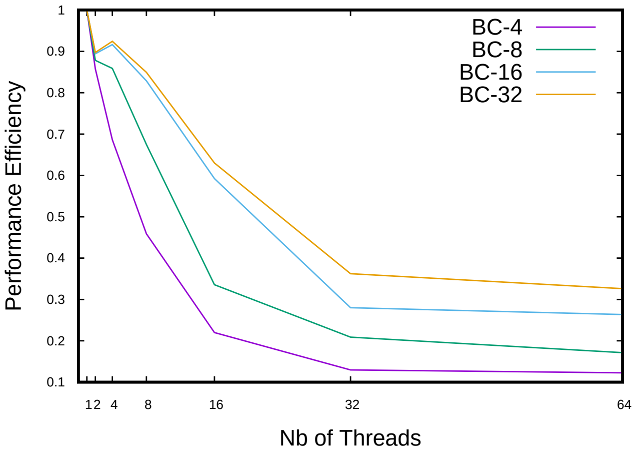

Fig. 14. Efficiency for the load based (BC) scenario and graph G_2

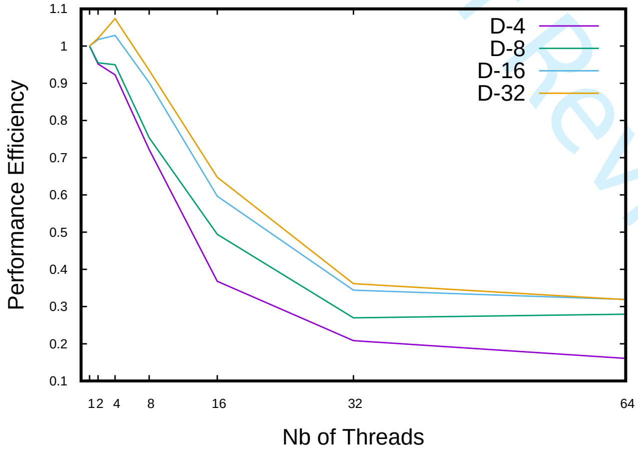

Fig. 12. Efficiency for the degree-based (D) scenario and graph G

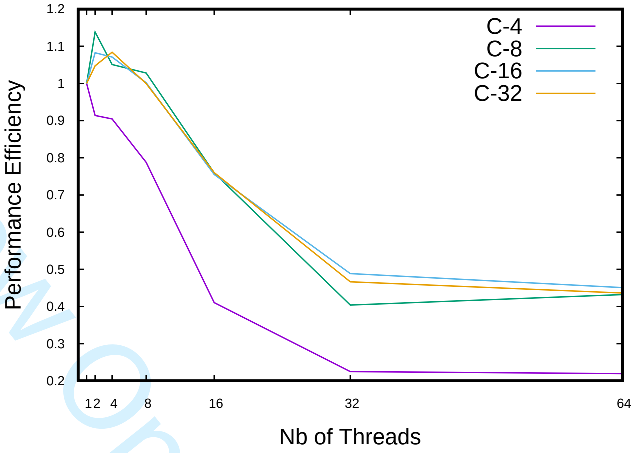

Fig. 15. Efficiency for the cascading (C) scenario and graph G_2

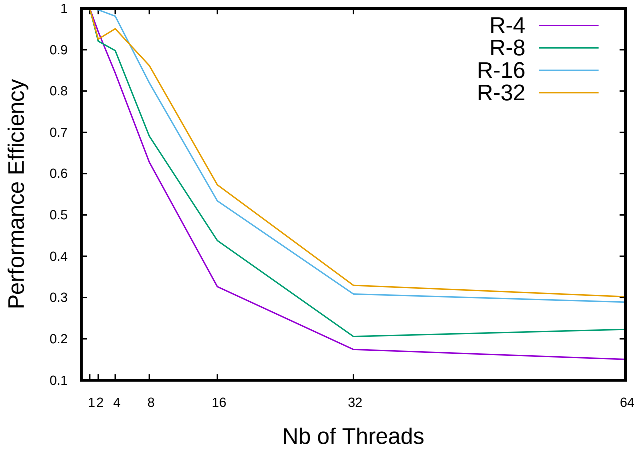

Fig. 13. Efficiency for the random (R) scenario and graph G

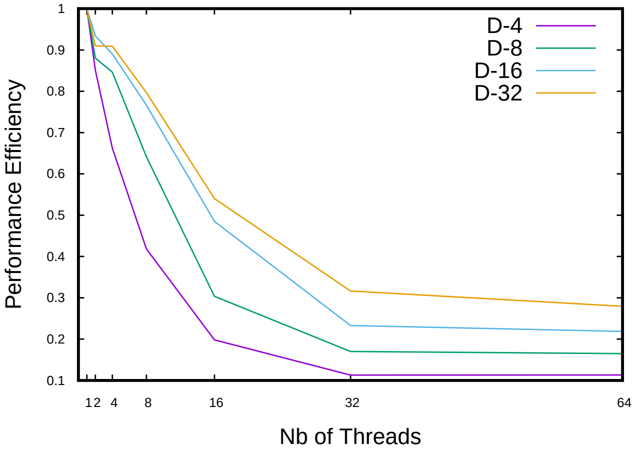

Fig. 16. Efficiency for the degree-based (D) scenario and graph G_2

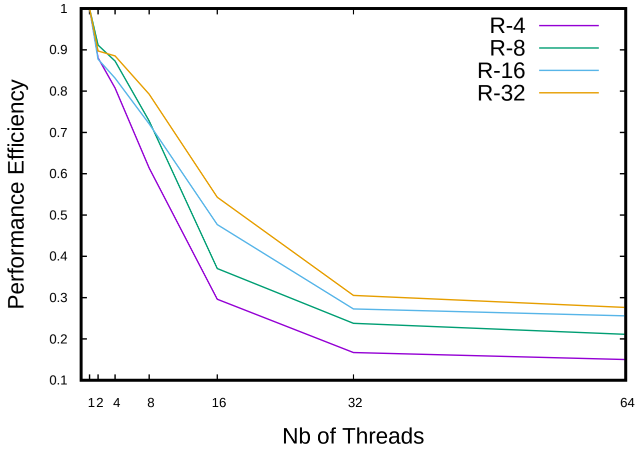

Fig. 17. Efficiency for the random (R) scenario and graph G_2

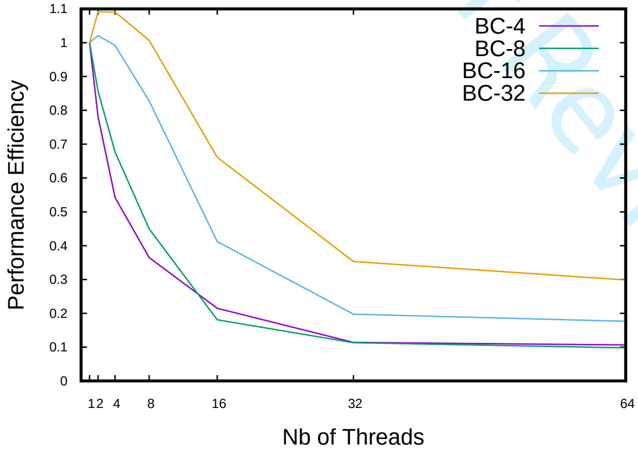

Fig. 18. Efficiency for the load based (BC) scenario and graph G_4

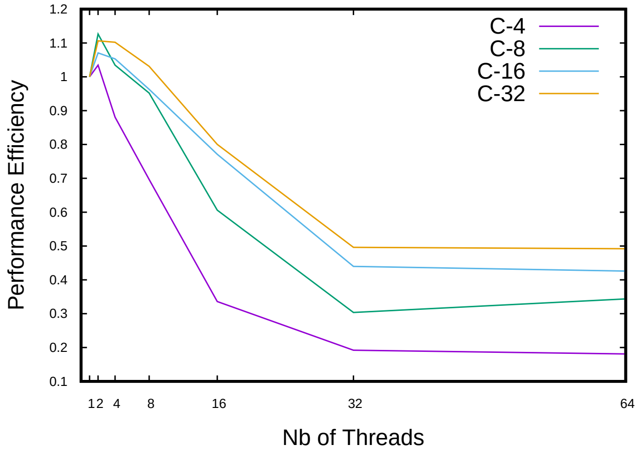

Fig. 19. Efficiency for the cascading (C) scenario and graph G_4

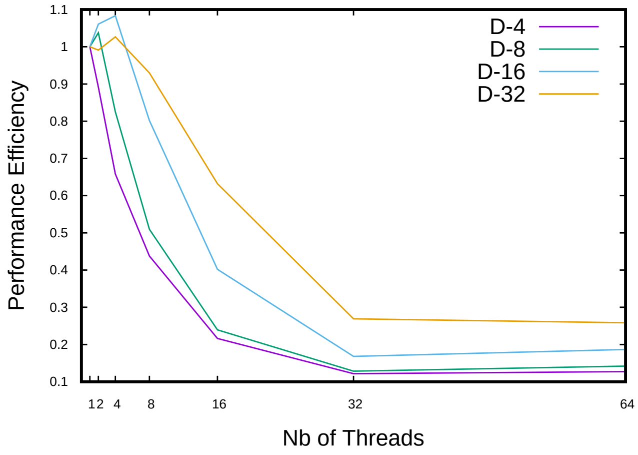

Fig. 20. Efficiency for the degree-based (D) scenario and graph G_4

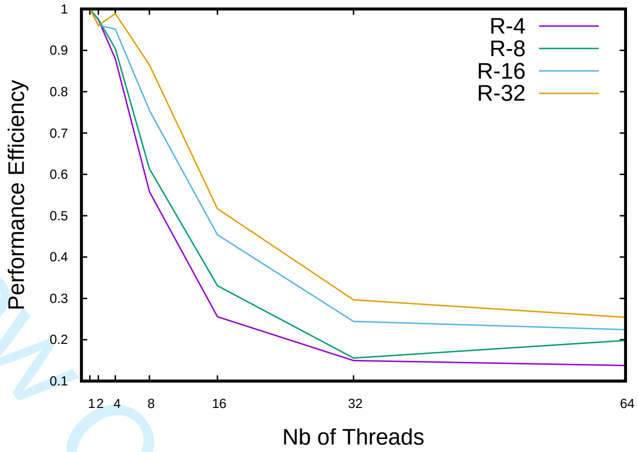

Fig. 21. Efficiency for the random (R) scenario and graph G_4

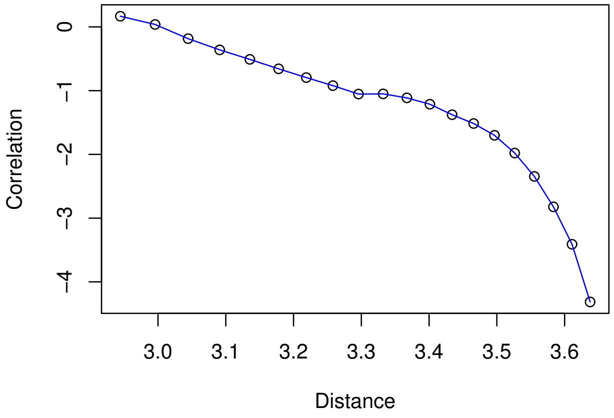

Fig. 22. Spatial correlation between nodes in the cascading scenario on a logarithmic scale.

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 synchronisation among threads. Our analysis exploits a high level of decentralisation 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 our contingency analysis software provide for all of the following:

1) Usability: The targeted audience of the system includes governmental and other vital organisations. 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, telephony.
2) Portability: The system can run on any Hadoop configured machine, which can be a serial machine, a multicore, 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 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 Review*, 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. 12 187–12 210, Oct. 2015.

[3] A. Bashan, Y. Berezin, S. V. Buldyrev, and S. Havlin, "The extreme vulnerability of interdependent spatially embedded networks," *Nature Publishing Group*, vol. 9, no. 9, pp. 1–6, 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," *Electric Power Systems Research*, vol. 81, no. 7, pp. 1334–1340, 2011.

[6] L. Dueñas-Osorio and S. M. Vemuru, "Cascading failures in complex infrastructure systems," *Structural Safety*, vol. 31, no. 2, pp. 157–167, Mar. 2009.

[7] D. Manik, M. Rohden, H. Ronellenfitsch, X. Zhang, S. Hallerberg, D. Witthaut, and M. Timme, "Network susceptibilities: theory and applications," *arXiv.org*, p. arXiv:1609.04310, Sep. 2016.

[8] R. Cohen, K. Erez, D. ben Avraham, and S. Havlin, "Breakdown of the Internet under Intentional Attack," *Physical Review Letters*, 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," *I. J. Bifurcation and Chaos*, 2007.

[10] E. Bompard, R. Napoli, and F. Xue, "Analysis of structural vulnerabilities in power transmission grids," *International Journal of Critical Infrastructure Protection*, vol. 2, no. 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," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 110, no. 30, pp. 12 159–12 159, Jul. 2013.

[12] L. Daqing, J. Yinan, K. Rui, and S. Havlin, "Spatial correlation analysis of cascading failures: Congestions and Blackouts," *Nature Scientific Reports*, vol. 4, pp. 1–6, Jun. 2014.

[13] R. Albert, I. Albert, and G. L. Nakarado, "Structural vulnerability of the North American power grid," *Physical Review E*, vol. 69, no. 2, pp. 025 103–4, 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," *Safety Science*, vol. 49, no. 6, pp. 807–812, Jul. 2011.

[15] R. V. Solé, M. Rosas-Casals, B. Corominas-Murtra, and S. Valverde, "Robustness of the European power grids under intentional attack," *Physical Review E*, vol. 77, no. 2, pp. 026 102–7, Feb. 2008.

[16] G. C. Ejebe and B. F. Wollenberg, "Automatic contingency selection," *IEEE Trans. on Power Apparatus and Systems*, vol. PAS-98, no. 1, pp. 92–104, 1979.

[17] A. O. Ekwue, "A review of automatic contingency selection algorithms for on-line security analysis," in *Proceedings of 3rd International Conference on Power System Monitoring and Control*, 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 *24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010, Atlanta, Georgia, USA, 19-23 April 2010 - Conference Proceedings*, 2010, pp. 1–7. [Online]. Available: 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," *Scientific Reports*, vol. 4, p. 5381, 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, Aug. 1990. [Online]. Available: http://doi.acm.org/10.1145/79173.79181

[21] R. Albert, H. Jeong, and A.-L. Barabási, "Error and attack tolerance of complex networks," *Nature (London)*, vol. 406, p. 378, 2000.

[22] R. Cohen, D. ben Avraham, and S. Havlin, "Resilience of the Internet to Random Breakdowns," *Phys. Rev. Lett.*, vol. 85, p. 4626, 2000.

[23] ——, "Breakdown of the Internet under intentional attack," *Phys. Rev. Lett.*, vol. 86, p. 3682, 2001.

[24] D. Callaway, M. Newman, S. Strogatz, and D. Watts, "Network Robustness and Fragility: Percolation on Random Graphs," *Phys. Rev. Lett.*, vol. 85, p. 5468, 2000.

[25] P. Hines and S. Blumsack, "A Centrality Measure for Electrical Networks," in *2008 The 41st Annual Hawaii International Conference on System Sciences*. IEEE, Sep. 2007, pp. 185–185.

[26] Z. Wang, A. Scaglione, and R. J. Thomas, "Electrical centrality measures for electric power grid vulnerability analysis," in *Proceedings of the 49th IEEE Conference on Decision and Control, CDC 2010, December 15-17, 2010, Atlanta, Georgia, USA*, 2010, pp. 5792–5797. [Online]. Available: https://doi.org/10.1109/CDC.2010.5717964

[27] G. Bianconi and F. Radicchi, "Percolation in real multiplex networks," *Physical Review E*, vol. 94, no. 6, pp. 060 301–5, Dec. 2016.

[28] D. S. Callaway, M. Newman, S. H. Strogatz, D. W. P. r. letters, and 2000, "Network robustness and fragility: Percolation on random graphs," *APS*.

[29] B. Karrer, M. E. J. Newman, and L. Zdeborová, "Percolation on Sparse Networks," *Physical Review Letters*, vol. 113, no. 20, pp. 211–5, Nov. 2014.

[30] F. Radicchi, "Percolation in real interdependent networks," *Nature Physics*, vol. 11, no. 7, pp. 597–602, Jun. 2015.

[31] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proc. Natl. Acad. Sci. USA*, vol. 99, no. 12, pp. 7821–7826, 2002.

[32] A. Cavagna, A. Cimarelli, I. Giardina, G. Parisi, R. Santagati, F. Stefanini, and M. Viale, "Scale-free correlations in starling flocks," *Proc. Natl. Acad. Sci.*, vol. 107, pp. 11 865–11 870, 2010.

[33] H. A. Makse, S. Havlin, and H. E. Stanley, "Modeling urban growth patterns," *Nature*, vol. 377, pp. 608–612, 1995.

[34] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Comm. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[35] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *in Proceedings of SIGMOD 2010*, 2010, pp. 135–146.

[36] R. H. Bisseling, *Parallel Scientific Computation*. Oxford University Press, 2004.

[37] J. M. D. Hill, W. F. McColl, D. C. Stefanescu, M. W. Goudrea, K. Lang, S. B. Rao, T. Suel, T. Tsantilas, and R. H. Bisseling, "Bsplib: The bsp programming library," *Parallel Computing*, vol. 24, pp. 1947–1980, 1998.

[38] L. G. Valiant, “A bridging model for parallel computation,” *Communications of the ACM*, vol. 33, pp. 103–111, 1990.

[39] 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 *in Proceedings of 11th USENIX Symposium on Operating Systems Design and Implementation, OSDI ’14, Broomfield, CO, USA, October 6-8, 2014.*, 2014, pp. 599–613. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/gonzalez

[40] P. Boldi and S. Vigna, “Axioms for Centrality,” *Internet Mathematics*, vol. 10, no. 3-4, pp. 222–262, 2014.

[41] U. Brandes, “A faster algorithm for betweenness centrality,” *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.