GridSpice: A Distributed Simulation Platform for the Smart Grid

Kyle Anderson, Jimmy Du, Amit Narayan, and Abbas El Gamal, *Fellow, IEEE*

***Abstract*—This paper describes GridSpice, a scalable open-source simulation framework for modeling, designing, and planning of the smart grid. GridSpice seamlessly integrates existing electric power simulation tools to enable modeling of large electric networks that blur the boundaries between generation, transmission, distribution, and markets. This is achieved via a cloud-based architecture that allows for parallelizing large simulation jobs across many virtual machines using a pay-as-you-go model. GridSpice simulations can be managed through a Representational State Transfer (REST) application programming interface (API), or through a Python library, allowing users to run simulations programmatically and interface with disparate data inputs, energy management systems (EMS), distribution management systems (DMS), and postprocessing tools. These capabilities make GridSpice an ideal tool for the development and testing of new grid control and optimization algorithms. GridSpice also provides an easy-to-use browser-based interface to allow novice users to begin without any setup or configuration on their local PC. A first implementation of the GridSpice framework integrates Gridlab-D and MATPOWER as simulation tools, and has been used for projects including optimizing the placement of distributed generation and developing optimal dispatch schedules for flexible loads. The GridSpice framework and Gridlab-D are freely available in open-source under the BSD license.**

***Index Terms*—Electric vehicles, multiagent systems, power system simulation.**

I. Introduction

THE ELECTRIC power grid, comprising utility companies, power system operators, market players, and other agents, is undergoing rapid change. Centralized generation is being complemented with renewable energy sources and storage systems. A prevalence of electric vehicles, unexpected consumer reaction to demand response programs, and distributed energy resources (DER) all add further stresses on an aging grid architecture. To deal with these changes, grid operators should become more proactive about replacing outdated components with more technologically advanced ones, adding sensing devices, such as smart meters, and using sophisticated distributed control systems to accommodate these changes. There is a high cost associated with reforming an asset such as the electric power system, a cost measured not only in dollars but also in terms of power disruption due to unintended consequences of upgrading large portions of the grid. These high costs can be somewhat ameliorated by simulation—modeling the grid as accurately as possible and using these models to develop and implement optimized control systems. This task is becoming more difficult, however, due to the increasing interdependencies among generation, transmission, distribution, and end-use loads. Existing electric power simulators provide well-proven point tools for transmission networks, e.g., Siemens PSS/E, distribution networks, e.g., Gridlab-D [3], OpenDSS [12], or general optimal power flow, e.g., MATPOWER [5]. Other simulation tools designed to study transients [14] are also important for both transmission and distribution system analysis, but are often designed as standalone solvers without detailed models of smart grid elements or models of customer behavior. Several frameworks have been proposed for tightly coupled cosimulation of communication systems and transmission networks based on the high-level architecture (HLA) and system-in-the-loop (SITL) [23] standards, but these do not support the cosimulation of transmission and distribution systems.

The integrated retail and wholesale (IRW) project at Iowa State University [13] provides a cosimulation testbed, but does not directly support parallelizing simulation jobs across a large cluster and does not provide a generic interface for incorporating new simulation tools or interfacing with disparate data sources. Their testbed runs on standalone workstations, limiting the scope of smart grid scenarios it can model.

In this paper, which provides a more detailed description of the work presented in [2], we describe GridSpice, a cloud-based simulation platform that addresses the aforementioned limitations of existing simulation systems. GridSpice provides a flexible framework that runs industry standard simulation tools as separate but synchronized processes on a cluster. Each subsystem within the network model runs in a simulator designed for that purpose while GridSpice synchronizes the boundary state of these loosely coupled processes. Since GridSpice can run each of these processes on a separate node of a cluster, it is possible to simulate a transmission network with hundreds of connected generators and distribution networks on a sufficiently large cluster.

In addition to partitioning large interdependent networks to run on a cluster, GridSpice also makes it easy to run

Manuscript received September 09, 2013; revised February 27, 2014 and May 03, 2014; accepted May 31, 2014. Date of publication June 30, 2014; date of current version November 04, 2014. This work was supported in part by the TomKat Center for Sustainable Energy, in part by Cisco Systems through the Energy and Environment Affiliate Program, in part by the Stanford Graduate Fellowship (SGF), and in part by the Stanford Electrical Engineering Research Experience for Undergraduates (REU) Program. Paper no. TII-13-0737.

K. Anderson, J. Du, and A. E. Gamal are with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: kyle.anderson@stanford.edu; jimmydu@stanford.edu; abbas@ee.stanford.edu).

A. Narayan is with the AutoGrid, Redwood Shores, CA 94065 USA (e-mail: amit@auto-grid.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2014.2332115

embarrassingly parallel tasks such as iterative grid analysis. Users can quickly evaluate the effects of many potential changes to the grid and compare the results. An example use case would be determining the ideal locations to add storage elements on the grid (i.e., the user evaluates each potential location independently in parallel).

The GridSpice framework allows users to edit models and control simulations through a Secure Representational State Transfer (REST) application programming interface (API). Since the REST interface is based on hypertext transfer protocol (HTTP) requests, users may control the system through the language of their choice and automatically synchronize their models with energy management systems (EMS) and distribution management systems (DMS). For user convenience, the client side of this REST API has been implemented in Python as a scripting tool to perform tasks such as iterative grid architecture optimization.

GridSpice eases adoption into existing work flows by providing an easy-to-use browser-based graphical user interface (GUI) with features including a geographical information system (GIS) editor, project explorer, object editor, and a wizard for importing projects from other systems. Since the GUI runs in the browser, it is platform-independent and does not require any setup on a user's workstation. New users can become familiar with the features of the system through the GUI before using the scripting interface, and advanced users can use the GUI to complement the scripting interface when they wish to perform visual checks on their models. This makes GridSpice ideal for both academic courses and professional use.

This paper is organized as follows. In Section II, we provide an overview of how we split a simulation into a set of loosely coupled processes running on a cluster with synchronized boundary state. In Section III, we provide some simple pseudocode examples of how to perform a simulation using GridSpice. In Section IV, we compare the GridSpice framework to existing cosimulation frameworks and analyze its performance. In Section V, we describe the software system implementation.

II. Simulation Clusters

GridSpice simulations run on a dynamically sized cluster consisting of a master node and worker nodes. The master node accepts simulation requests from the front-end server as described in Section V, and starts a supervisor process for each new simulation. The supervisor process is responsible for starting the subsimulations, which run on the worker nodes, and keeping the shared state synchronized. The supervisor process adds these tasks to an Oracle GridEngine queue that assigns the task to the worker node with the least load. Each worker node has a configurable number of slots, each of which can run one task. Fig. 1 depicts a cluster in which there are two simulations running. Each simulation has a single supervisor process. The first worker node is currently running four tasks, the second worker node is running two tasks, and the last worker node is running six tasks. The next time either simulation supervisor adds a new task, it is scheduled on the second worker node since it is the least loaded. The master node continuously reads the central processing unit (CPU) utilization on each

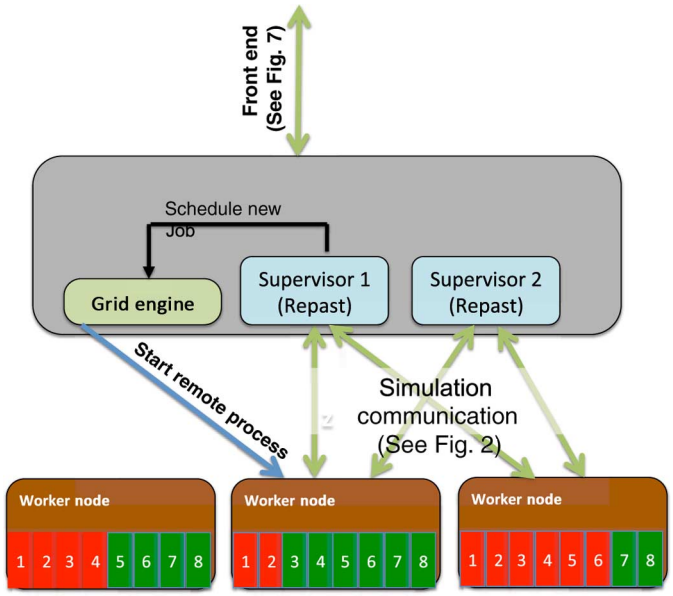

Fig. 1. Simulation cluster.

worker node. If all worker nodes reach their CPU bottleneck, the master node automatically starts up a new worker node. Conversely, if the master node detects worker nodes with no load, it shuts them down to conserve cloud resources.

The supervisor process creates a task for each network subsimulation. Each GridSpice simulation consists of exactly one transmission network, zero or more distribution networks, and zero or more generators. However, users can run distribution-only simulations by creating a single-bus transmission network with the distribution network(s) attached to it. Similarly, users can run transmission-only simulations by fixing the loads and the generator parameters for each bus instead of attaching a distribution network. Each network runs in a subsimulator as a remote task.

In addition to attaching remote subsimulators to each bus, the user can define a local agent within the supervisor process as shown in Fig. 2. This is useful when the bus uses simple logic such as reading from a data file or a given probability distribution. For example, a user may choose to simulate the NYISO 2935-bus transmission network [5] with 200 attached distribution simulation instances, 20 attached generation simulation instances, and 2715 local agents that define the behavior of the buses that do not have an attached simulation instance.

Once the remote tasks have started, the supervisor process uses the RePast Simphony [4] to maintain a global simulation clock and synchronize the boundary states between these subsimulations, as shown in Fig. 2. The supervisor maintains a proxy for each worker task, and keeps track of which worker tasks need update messages when a dependent worker task changes its state.

Each remote task runs a simulation tool within a wrapper to synchronize any shared state with its proxy in the supervisor. The proxy defines a list of input and output shared variables using Java annotations. The user can add a new dependency by adding the variable in the proxy class along with the appropriate input/output annotation. After adding a new dependency,

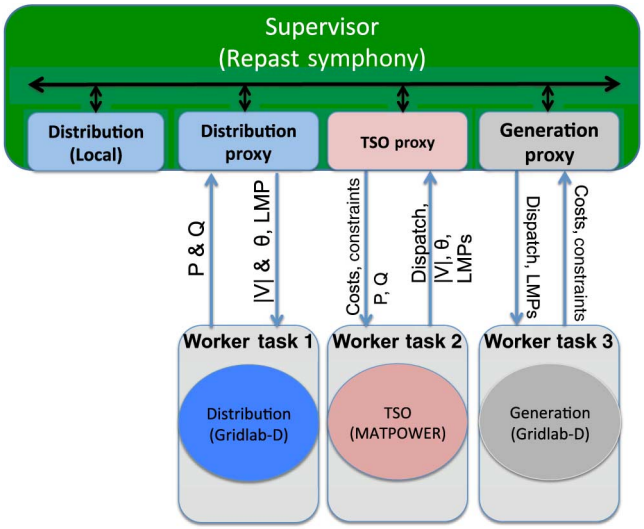

Fig. 2. Simulation consisting of three remote tasks and one local task.

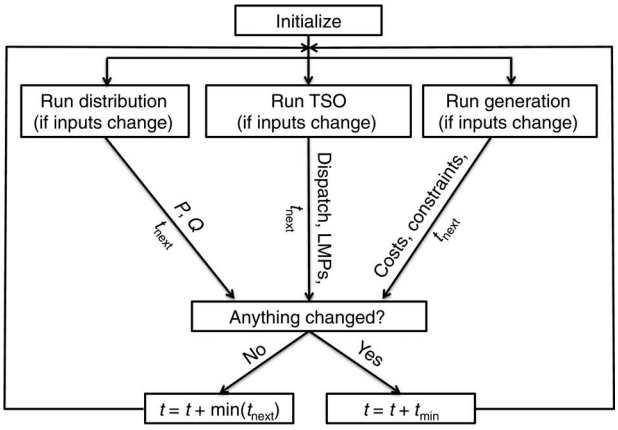

Fig. 3. Method for determining how to advance global simulation clock.

the wrapper must be adjusted to send or receive the new variable from the simulator.

The GridSpice framework uses Gridlab-D for distribution and generation simulation, which supports three-phase unbalanced power flow and provides numerous models for smart grid hardware, customer behavior, and generators. The agent acting as the system operator runs a lightweight transmission and economic dispatch package based on MATPOWER. In its original form, MATPOWER provides one-shot solutions of the optimal power flow problem. We have modified MATPOWER to rerun at each time step, resetting the ramping constraints for the next time step based on the operating points from the previous time step. This does not guarantee a globally optimal dispatch schedule over time, and we plan to improve this scheme using dynamic programming in a future release of GridSpice.

At each time step, the system operator task publishes the locational marginal price, voltage magnitude, and voltage angle for each load bus on the transmission network as well as the power injection, voltage magnitude, and voltage angle for each generation bus. The system operator task listens to updates of P and Q from each load bus as well as updates of the costs and constraints from each generator. Analogously, the distribution tasks publish their P and Q values at each time step, and listen to updates of their voltage magnitude, voltage angle, and LMP. The generation task publishes its costs and constraints as well as receive notifications about dispatch schedules, voltage magnitude, and voltage angle on the transmission grid.

The GridSpice supervisor is an agent-based discrete event simulator in which each subsimulation task is an agent. Each time a task updates its state, it decides the time $t_{\text{next}} \geq t_{\min}$ at which it needs to update again. However, the task has the opportunity to update before t_{next} but after $t_{\min}$ if any of its dependent inputs change, as illustrated in Fig. 3. For example, a distribution network may recompute P and Q when the parent LMP changes. However, it may not recompute its P and Q at time steps more granular than $t_{\min}$. Thus, $t_{\min}$ should be set to the minimum granularity of all dynamic control systems used in the models. The ramifications of adjusting $t_{\min}$ are discussed in greater detail in Section IV. The current simulators integrated within GridSpice are intended for analysis at intervals greater than one second.

GridSpice uses a pessimistic approach to advancing the simulation. The internal clock for each task does not advance to the next time step until the global clock maintained in the supervisor process has advanced. This approach helps make the GridSpice platform simulator-agnostic because simulators are not required to implement check pointing or rollback.

III. Examples and Applications

We first describe two simple examples that utilize the features of GridSpice and can serve as starting points for more-involved scenarios. The first example shows how GridSpice's parallel architecture can be used to perform iterative heuristic analysis of solar panel placement. The second example highlights the cosimulation capabilities of GridSpice by finding the optimal placement of solar panels across multiple distribution networks where the placement may impact locational marginal prices on the network. We then briefly describe several more involved applications that have used GridSpice.

A. Example 1 (Solar Panel Placement)

This example shows how to use GridSpice to place 200 new solar panels in a distribution network using a heuristic. The transmission network is modeled as a single bus. All customers are subject to the same locational marginal prices, hence the objective is based solely on network losses, operating violations, and the peak-shaving behavior of solar output. The pseudocode for determining a placement is shown in Algorithm 1. In each round, the algorithm places a single solar panel by randomly selecting 50 potential locations, testing the impact of adding a solar panel at each of those locations, and finally placing the new solar panel at the location with the best score. This procedure continues until all 200 panels have been placed (for a total of 200×50 simulation runs). This simple algorithm is only meant to demonstrate GridSpice's capability

Algorithm 1. Greedily place solar panels on a single distribution network

```
Model ← loadModel()
for k = 1 to 200 do
    candidates ←{}
    for j = 1 to 50 do
        model′ ←Model.copy()
        distNetwork ←model'.getDistribution(0);
        size ←distNetwork.getCustomers().size();
        rand ←Random( 0, size )
        customer ←distNetwork.getCustomer(rand);
        customer.attach( new SolarPanel() )
        candidates ←(model′∨{candidates})
    end for
    Model ← argmax_{C∈candidates} Score(parallelSim(C))
end for
```

to create parallel simulations. More sophisticated algorithms may use additional heuristics to select potential locations.

B. Example 2 (Cosimulation)

This example extends the previous example to highlight the cosimulation capabilities of GridSpice. Instead of using a single bus transmission network as in Example 1, this example uses the IEEE 14-bus test model [11]. The generators and distribution networks attached to the transmission buses are summarized in Table I.

As in Example 1, we use a simple greedy algorithm to select customer locations. However, the customer locations can now be spread across different distribution networks, which may be attached to different transmission buses. Therefore, the locational marginal prices of the transmission network can now affect the assignment of the panels. The updated pseudocode is shown in Algorithm 2.

C. Applications

The GridSpice framework has been used in a study on demand response and in several class projects.

1) *Simulating integrated volt/var control and distributed demand response* [1]: This paper proposes a new integrated volt/var control scheme that uses demand response capacity to improve the reliability and reduce power consumption of a distribution network. GridSpice was used to test the control scheme outlined in [17].
2) *Stochastic control of electric vehicle charging* [10]: In this project, GridSpice was used to show the efficacy of a multiagent reinforcement learning system for electric vehicle charging on a constrained network using methods outlined in [18].
3) *Comparison of community-scale and distributed residential-scale photovoltaics* [7]: In this class project, GridSpice was used to compare various placements of residential rooftop solar panels. This study shows that solar panels and active inverters can be used to improve power quality throughout the distribution grid.

TABLE I
TRANSMISSION NETWORK FOR EXAMPLE 2

Bus	Type	Connections
0	Swing	Generator 1
1	PV	Generator 2
3	PV	Generator 3
4	PQ	Distribution A
5	PQ	Distribution B , Distribution C
6	PV	Generator 4
7	PQ	Distribution D , Distribution E
8	PV	Generator 5
9	PQ	Distribution F, Distribution G, Distribution H
10	PQ	Distribution I
11	PQ	Distribution J
12	PQ	Distribution K
13	PQ	Distribution L
14	PQ	Distribution M

Algorithm 2. Greedily place solar panels across multiple distribution networks considering varying locational marginal prices

```
Model ← loadModel()
for k = 1 to 200 do
    candidates ←{}
    for j = 1 to 50 do
        model′ ←Model.copy()
        networkCount ←model'.getDistributions().size();
        x ←Random( 0, networkCount )
        customerCount ←model'.getDistNetwork(x).size();
        x ←Random( 0, customerCount )
        customer ←model'.getNetwork(0).getCustomer(x);
        customer.attach( new SolarPanel() )
        candidates ←(model′∨{candidates})
    end for
    Model ← argmax_{C∈candidates} Score(parallelSim(C))
end for
```

4) *Photovoltaic integration on distribution networks* [9]: This study shows the benefits of adding both storage and rooftop solar at residential customer locations. GridSpice was used to estimate the ideal amount of storage under various assumptions about storage costs, electricity prices, and feed in tariffs.
5) *Electrical characteristics of distributed photovoltaics* [8]: In this class project, GridSpice is used to characterize the dangers arising from back feeding on residential solar. The project estimates the maximum penetration levels under various safety tolerances and volt/var regulation schemes.

IV. ANALYSIS AND COMPARISON

This section compares GridSpice to existing cosimulation frameworks in terms of scalability, flexibility, scope, and time-granularity.

A. Scalability

The primary advantage of GridSpice over existing cosimulation platforms is scalability. Although cosimulation frameworks support distributed simulation, they do not provide direct support for running on a cloud infrastructure. For example, electric power and communication synchronizing simulator (EPOCHS) [26] is designed to run on dedicated hardware using static provisioning. In contrast, GridSpice uses Amazon Web Services (AWS) and creates dynamically sized clusters, adding and deleting virtual machines as needed.

Aside from the obvious cost and scalability benefits of running in the cloud, GridSpice provides benefits over existing distributed cosimulation platforms by using a hierarchical approach to dividing tasks. In [23], several cosimulation methods based on the HLA [22] standard are discussed. The HLA architecture uses a run-time infrastructure (RTI), which handles the global simulation clock and exchanges boundary state between subsimulations. In that sense, the RTI is very similar to the GridSpice supervisor. However, unlike HLA, GridSpice allows light-weight agents to run locally within the supervisor. This improves scalability by removing unnecessary messaging overhead for simple agents, as discussed in Section II.

GridSpice is highly scalable since its integrated simulators are CPU-bound and "loosely coupled," meaning that the messaging overhead from interactions between simulators is insignificant compared to the CPU overhead of running each simulator. This property is evident in Fig. 4, which shows the time required to simulate the NYISO 2935-bus transmission network [5] with different numbers of connected distribution feeders for a 24-h period using alternating current optimal power flow (AC OPF) on the transmission network. Instances of each of the feeders shown in Fig. 4 are connected to randomly chosen *PQ* buses from the transmission network. Customer load data are generated in 15-min time intervals. Thus, the power flow for the transmission network and each distribution network is solved 96 times over the course of the simulation. Since each of these simulations is loosely coupled, our results show that simulations achieve nearly linear speedup by provisioning additional virtual machines until the point where each network is running on its own virtual machine.

The GridSpice framework does not add significant overhead to synchronize the boundary state between subsimulations running on different nodes. Fig. 5 shows that on average a virtual machine involved in the simulation spends about 70% of its time executing simulation related CPU instructions, 20% waiting on disk I/O for data such as the network model or load information, and 10% synchronizing messages to keep the boundary state consistent.

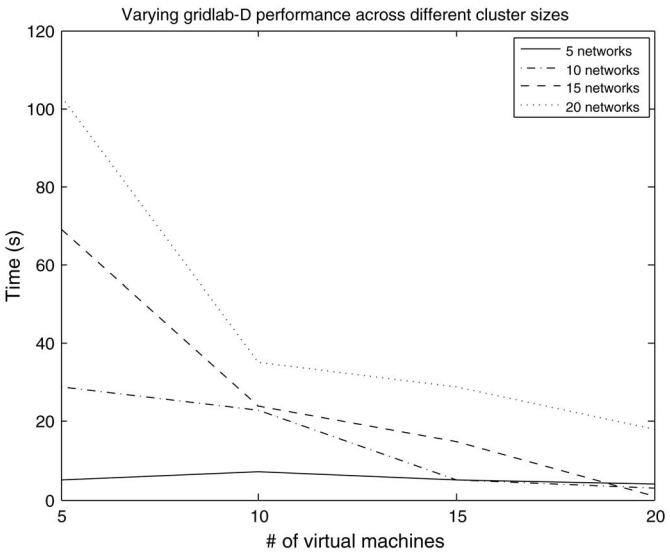

Fig. 4. Time required to simulate the NYISO 2935-bus transmission network with different numbers of attached distribution networks and available virtual machines.

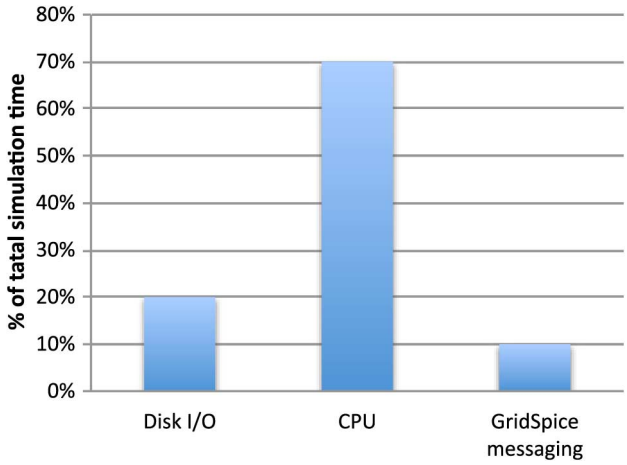

Fig. 5. Performance measurements of the NYISO 2935-bus transmission network with 100 synchronized distribution networks running on 100-VMs.

B. Flexibility

Since existing power system simulation tools are often designed to run on specific computing platforms, there has been considerable work on developing interoperable methods for synchronizing tools running on different platforms. Notably, the HLA [22] standard defines a mechanism for federating simulation tools using an RTI to synchronize boundary state. GridSpice uses an approach similar to HLA, exchanging synchronization messages over a network. GridSpice provides this flexibility in a particularly cost-effective way because it can modify both the number and size of the instances of each platform depending on the requirements of the current simulation. For example, if the current simulation has many subsimulations that can run only on a Windows machine, GridSpice can dynamically create and destroy the necessary number of instances. AWS provides support to run many flavors of Windows and Linux virtual machines and to created customized images for a simulator. When integrating a new simulation tool into GridSpice, the user specifies the name of the machine image in the supervisor. The supervisor then ensures that the associated tasks are assigned to an appropriate machine, as shown in Fig. 6.

The GridSpice framework offers further flexibility by providing a template for the supervisor proxy and for the simulator wrapper discussed in Section II. To add a new simulator,

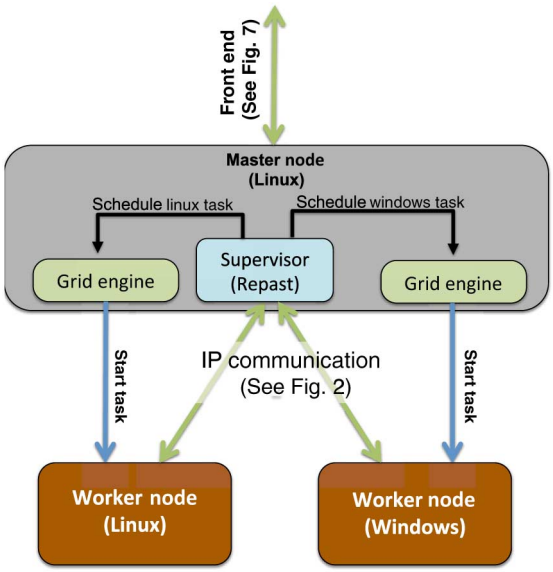

Fig. 6. Coordination between cross-platform simulation tasks.

the user modifies the template to include the shared state variables. One benefit of using RePast is that the user adds simple Java annotations to specify the input and output dependencies, and the routing logic within the supervisor relies on a widely adopted and robust code. The wrapper template implements the functionality to send and receive the shared variables on the worker node. The user is responsible for extending the wrapper template to interface with the new simulator.

C. Scope

In theory, any cosimulation framework provides complete modeling scope if it allows the integration any simulator. In practice, frameworks are most useful if they provide direct and tested support for a given subsimulator. Most existing frameworks provide out-of-the-box support for cosimulating communication networks and transmission networks, e.g., [24]–[29], [31]. These frameworks are useful for studying problems such as protection on transmission systems, but do not provide robust support or a large array of models for distribution systems. A few frameworks, e.g., [19]–[21], provide direct support for distribution modeling, but they do not provide the scalability and flexibility benefits discussed above. GridSpice, through its integration with Gridlab-D, provides support for detailed modeling of end use loads and customer behavior alongside transmission and economic dispatch.

D. Time-Granularity

Many existing cosimulation platforms, e.g., [16], [24]–[29], [31], focus on fine-grained (millisecond-level) cosimulation of communication networks and power networks. GridSpice is designed to provide courser grained synchronization across loosely coupled transmission and distribution subsimulators. Subsecond interactions relating to communications and control systems should be fully contained within the integrated subsimulators running on a single machine. The recent release of Gridlab-D 3.0 already provides capabilities for modeling communications systems and efforts for integrating the more robust ns-3 simulator [30] are already in progress. Nonetheless, GridSpice allows the user to define the shared state and synchronization intervals between subsimulators. Thus, the granularity is ultimately up to the user.

TABLE II
GRIDSPICE MAXIMUM UPDATES PER SECOND

Remote tasks	Single	$\sqrt{N}$	All
10	10 000	3100	1000
50	2000	280	40
100	1000	100	10
200	500	35	3

Table II provides the maximum update rate of the GridSpice supervisor, which can be used to estimate the feasibility of integrating a certain set of subsimulators. In a simulation with 200 remote tasks (subsimulators) in which updates to the boundary state from each remote task affect an average of $\sqrt{200}$ other remote tasks, the GridSpice framework can handle up to 35 updates per second. Each boundary state variable counts as a separate update. Thus, if a remote task has five boundary state variables, it should update these shared variables less than seven times per second. If the update rate exceeds this rate, the virtual time clock advances slower than the realtime clock.

V. SYSTEM ARCHITECTURE AND IMPLEMENTATION

This section provides an overview of the architecture of the GridSpice system and implementation details of its user interface, model storage, scripting, and interfacing with external tools and data inputs. Fig. 7 gives a top-level view of the entire system. The front-end server coordinates all actions between the user, the data, and the simulators. There are three different mechanisms through which the user can interact with the front-end server—a browser-based GUI, a Python library, and any third-party library using the REST interface. The front-end server begins a simulation by sending a request to the master node of a simulation cluster, beginning the process described in Section II. We describe each component of this system in further detail below.

A. Simulation Clusters

The GridSpice system can have many simulation clusters of varying sizes, and permissions for each cluster can be set on a per-user basis. The operation of the simulation clusters depicted in Fig. 7(a) is explained in detail in Section II.

B. Front-End Server

The front-end server coordinates all the actions between the user, the data, and the simulators as shown in Fig. 7(b). The components of the front-end server are expanded in Fig. 8 and discussed below.

The front-end server performs the following key functions.

1) *Importing models.* The front-end server can import models from a CIM-based XML format or the Gridlab-D

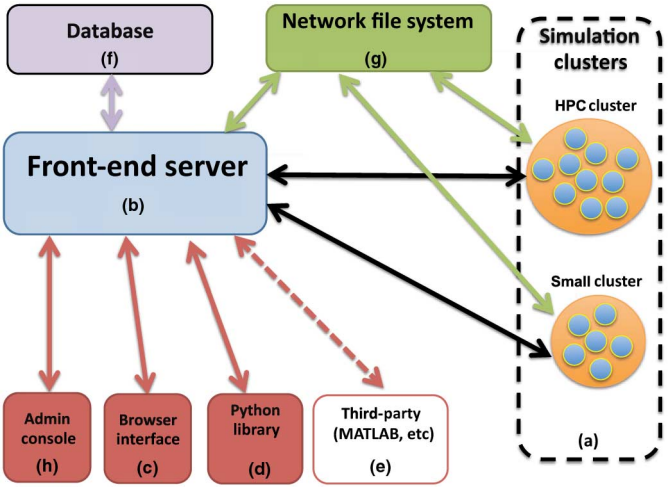

Fig. 7. GridSpice top-level system architecture. (a) Simulation clusters. (b) Front-end server (c) Browser interface. (d) Python library. (e) Third-party interfaces. (f) Database. (g) Network file system. GridSpice top-level system architecture.

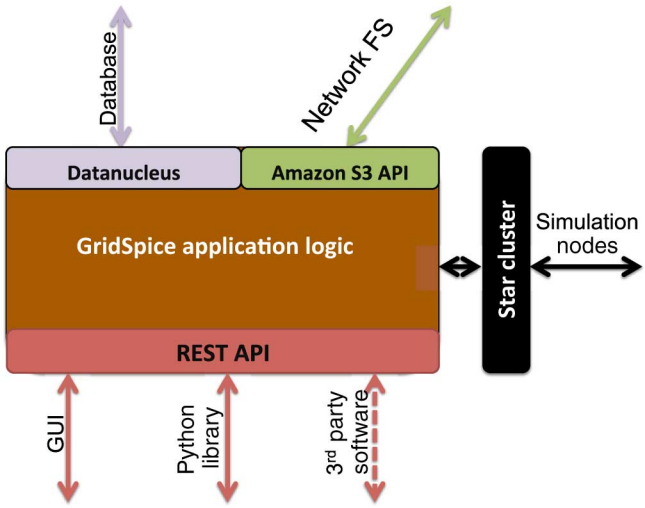

Fig. 8. Front-end server.

GLM format. These models can come with or without GIS data. If the models are provided with GIS information, the browser interface provides a map-view of the network in the graphical editor. If the models do not contain GIS information, we use an algorithm derived from the Java Universal Network Graph Framework [6] to layout the network on a planar graph with minimal line crossings. This algorithm uses a simulated annealing technique where the penalty function is based on a repulsion factor for each pair of nodes, an attraction factor for each pair of connected nodes, and a penalty for line crossings. The attraction factor is inversely proportional to the length of the lines (stronger attraction factors for nodes connected by shorter lines). In the beginning, the algorithm makes large movements to discover node placements with few line crossings. After several iterations, the size of the movements slow, and algorithm continues to adjust line lengths to find a reasonable visualization of the network.

2) *User authentication.* When using the Python library or a third-party library, the user must include an API key that will authenticate each request. The user can optionally reset the API Key at any time, allowing the user to give temporary access to his GridSpice resources to another entity such as an external data input or a postprocessing tool (MATLAB, etc.). When using the browser GUI, the user must login to the front-end server, and a browser cookie will be set to authenticate all future requests.
3) *Access control lists.* The front-end server maintains an access control list for each model and ensures the user has permission to perform the requested action based on his cookie or API key.
4) *Simulation initialization and monitoring.* The front-end server is responsible for sending the control message to the master node in the simulation cluster that starts the simulation supervisor. This control message indicates the location of the model files on the network file system. The front-end server logs heartbeat messages from the simulation cluster regarding the progress of the simulation.
5) *Cluster management.* Users with appropriate privileges can start, stop, create, and delete clusters of varying sizes. Since cloud services such as AWS charge by the hour, this can be used to start a new large cluster for a short period of time without incurring excessive costs.

In the current version of GridSpice, there is only one front-end server that fulfills these roles for all users and simulations. However, the architecture could be extended to include multiple front-end servers along with a load balancer.

C. Browser Interface

GridSpice provides a GUI that allows users to perform the basic tasks of the GridSpice system. The GUI provides a subset of the features available through the REST API and Python library. It is intended to be a tool for beginners to familiarize themselves with the system and for experienced users to sanity check their models. The interface runs in JavaScript in the client's browser and was built using Google Web Toolkit and the EXT-GWT library.

The GUI provides several different useful views.

1) *GIS editor.* If the network model contains GIS information, the GIS Editor allows users to visualize the network on a Google map as shown in Fig. 9. If the network model does not contain GIS information, the GIS Editor can visualize the map on a blank background after running the GridSpice layout tool described in Section V-B. The user can add or delete objects in the map view, and also click objects to open the corresponding properties editor. The map view layers objects hierarchically and adjusts the number of displayed items for large networks.
2) *Explorer editor.* The GUI offers a separate explorer view that allows the user to list elements in a hierarchical

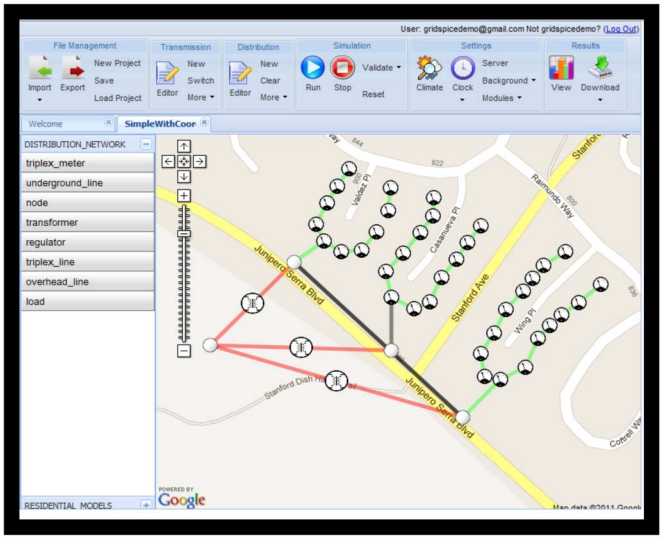

Fig. 9. GIS Editor in GridSpice browser UI.

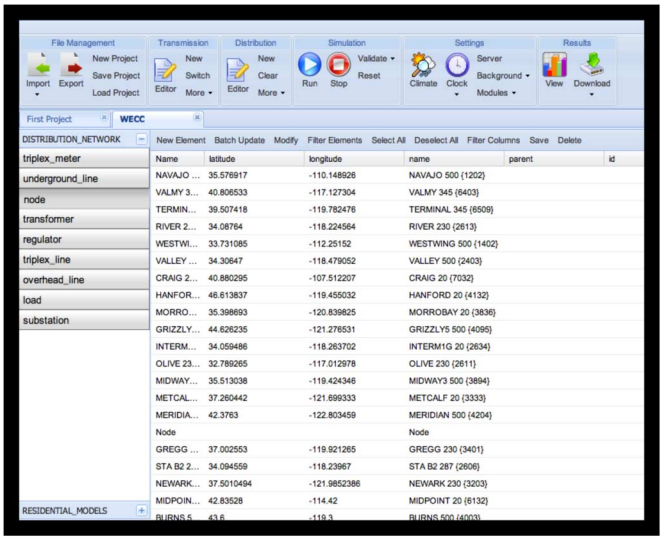

Fig. 10. Hierarchical Explorer Editor in GridSpice browser UI.

spreadsheet format shown in Fig. 10. In this view, the user can apply batch updates to properties of objects that match a given regular expression. The user can apply a number of built-in macros such as adding a roof-top solar to 50% of the buildings in the network.

3) *Object editor.* The object editor provides a graphical menu to edit the properties of the object. For example, a transformer object has a number of editable properties such as phase, max power, and nominal voltage. The property editor uses appropriate widgets for each different data type so the user understands the available options and knows which properties must be defined for the model to be valid. For example, a string or integer uses a validated textbox, a set uses checkboxes, and an enumerated type uses a drop-down menu. This view contains a description field to explain some of the esoteric properties.

4) *Import wizard.* The GUI provides an import wizard through which users can upload network models and load data files into the open project. The upload wizard can also accept a compressed zip archive to ease the import of a large projects containing many networks and load files.

D. Python Library

The Python library provides a convenient way for users to create and edit network models, run simulations, and collect results. The library provides a class for each of the available object types with their defined properties and units. This allows the user to easily understand how to change the models.

E. Third-Party Library

The GridSpice front-end server implements a REST interface with standard CRUD (create-read-update-delete) interface to each of the main entities in the system. This standardization allows the interface to be implemented in any language or external library. The provided Python library and Browser-based GUI are examples of client-side applications that use this interface, and can be referenced as a template for making calls to the GridSpice server from the language or library of choice. Since the REST interface uses the HTTP protocol, built-in MATLAB libraries can be used to authenticate with the application server and download the simulation outputs.

Furthermore, because of the stateless semantics of REST, there is no requirement that a single client implementation be used. For example, a user could use the Python library to setup simulations, use the browser GUI to perform a visual sanity check on models, and use MATLAB for postprocessing.

F. Database

The front-end server represents metadata associated with all accounts, projects, models, simulations, and data files as plain old java objects (POJOs). These objects are persisted and restored using the Java Data Objects API on the DataNucleus platform. The advantage of this strategy is that the application logic running on the front-end server is decoupled from the database implementation. An administrator may choose to use a different database depending on size and scale of a particular GridSpice installation. DataNucleus provides tools for easy integration with a number of relational databases such as MySQL as well as nonrelational databases such as HBase or Google BigTable. The current release of GridSpice uses a single instance MySQL server that meets our demands.

G. Network File System

The GridSpice network file system is implemented using Amazon S3, which provides a redundant data storage infrastructure that allows data to be securely read and written from anywhere on the web. This file system is used to store all loosely structured files in GridSpice system such network models, time-series load data, and simulation output and error logs.

H. Administration Console

GridSpice provides a web-based administration console hosted on the front-end server. This console allows an authorized administrator to create user accounts, grant users access to projects and models, create new simulation clusters, and start/stop/delete existing simulation clusters.

VI. Conclusion

GridSpice provides a scalable and extensible platform for modeling, designing, and planning of the smart grid. It allows utilities, energy services providers, regulators, researchers, educators, and students to solve problems in the smart grid that cannot otherwise be accurately modeled by existing simulators either because of scale or modeling capability. GridSpice's ability to run on public cloud systems with a pay-as-you-go model makes it possible for budget-constrained entities to simulate complex smart grid scenarios. Its generic simulator wrappers also allow users to plug in new tools to seamlessly interoperate with current tools. We have shown that the framework can run aggregated simulations using separate distribution and transmission system simulators, while adding minimal overhead.

Acknowledgment

The authors would like to thank B. Entriken with the Electric Power Research Institute (EPRI) and Prof. R. Rajagopal of Stanford for their insight and guidance. They would also like to thank Pacific Northwest National Labs for their support of the Gridlab-D package through this process, and the anonymous reviewers for suggestions that have greatly improved this paper.

Appendix A
Getting Started With GridSpice

The entire GridSpice system is available in open source under the BSD license. The code is available as a set of five repositories for the each GridSpice subsystem: front-end server, simulation supervisor, simulation node, Python API, and GUI. In order to ease the process of setting up a GridSpice cluster, we have created several scripts and publicly available Amazon Machine Images that allow the system to be automatically created within a user's Amazon EC2 account. This option provides a much easier way to create the system than building it from the source.

References

[1] K. Anderson and A. Narayan, "Simulating integrated volt/var control and distributed demand response using GridSpice," in *Proc. IEEE Conf. 1st Int. Workshop Smart Grid Model. Simul. (SGMS)*, Oct. 2011, pp. 84–89.

[2] K. Anderson, J. Du, A. Narayan, and A. E. Gamal, "GridSpice: A distributed simulation platform for the smart grid," in *Proc. Workshop Model. Simul. Cyber-Phys. Energy Syst. (MSCPES)*, May 20, 2013, pp. 1–5.

[3] D. P. Chassin and K. Schneider, "GridLAB-D: An open-source power systems modeling and simulation environment," in *Proc. IEEE Transmiss. Distrib. Conf. Expo.*, Chicago, IL, USA, Apr. 2008, pp. 1–5.

[4] M. J. North *et al.*, "Complex adaptive systems modeling with repast simphony," *Complex Adapt. Syst. Model.*, 2013, doi:10.1186/2194-3206-1-3.

[5] R. D. Zimmerman, C. E. Murillo-Snchez, and R. J. Thomas, "MATPOWER steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, Feb. 2011.

[6] J. Madadhain, D. Fisher, P. Smyth, S. White, Y. B. Boey, "Analysis and visualization of network data using JUNG," *J. Stat. Software*, pp. 1–25, 2005.

[7] K. Kallevig-Childers, E. Thomas, and L. Vogel, "Comparison of community-scale and distributed residential-scale photovoltaics," Stanford University, Stanford, CA, USA, CEE272R: Modern Power Systems Engineering Project Reports, Spring, 2012.

[8] C. Nolen, A. Pearson, and G. Provost, "Electrical characteristics of distributed photovoltaics," Stanford University, Stanford, CA, USA, CEE272R: Modern Power Systems Engineering Project Reports, Spring, 2012.

[9] E. Arnold, A. Burdick, and S. Mei, "Photovoltaic integration on distribution networks," Stanford University, Stanford, CA, USA, CEE272R: Modern Power Systems Engineering Project Reports, Spring, 2012.

[10] K. Anderson, "Stochastic control of electric vehicle charging," Stanford University, Stanford, CA, USA, CS229: Machine Learning Project Reports Autumn 2012.

[11] Power Systems Test Case Archive. (2012). University of Washington Electrical Engineering, Seattle, WA, USA [Online]. Available: http://www.ee.washington. edu/research/pstca/

[12] D. Montenegro, M. Hernandez, G. A. Ramos, "Real time OpenDSS framework for distribution systems simulation and analysis," in *Proc. 6th IEEE/PES Transmiss. Distrib.: Latin America Conf. Exposition (T&D-LA)*, Sep 3–5, 2012, pp. 1–5.

[13] D. Aliprantis, S. Penick, L. Tesfatsion, and Huan Zhao "Integrated retail and wholesale power system operation with smart-grid functionality," in *Proc. IEEE Power Energy Soc. Gen. Meet.*, Jul. 25–29, 2010, pp. 1–8.

[14] K. N. Miu and H.-D. Chiang, "Electric distribution system load capability: problem formulation, solution algorithm, and numerical results," *IEEE Trans. Power Del.*, vol. 15, no. 1, pp. 436–442, Aug. 2000.

[15] W. Chun-Yu *et al.*, A framework for multi-agent-based stock market simulation on parallel environment, pp. 561–570.

[16] C.-h. Yang, G. Zhabelova, C.-W. Yang, and V. Vyatkin, "Cosimulation environment for event-driven distributed controls of smart grid," *IEEE Trans Ind. Informat.*, vol. 9, no. 3, pp. 1423–1435, Aug. 2013.

[17] H. Ma, K. W. Chan, and M. Liu, "An intelligent control scheme to support voltage of smart power systems," *IEEE Trans. Ind. Informat.*, vol. 9, no. 3, pp. 1405–1414, Aug. 2013.

[18] F. Kennel, D. Gorges, and S. Liu, "Energy management for smart grids with electric vehicles based on hierarchical MPC," *IEEE Trans. Ind. Informat.*, vol. 9, no. 3, pp. 1528–1537, Aug. 2013.

[19] A. Monti, M. Colciago, P. Conti, M. Maglio, and R. Dougal, "A co-simulation approach for analysing the impact of the communication infrastructure in power system control," in *Proc. Grand Challenges Model. & Simul. Conf. (GCMS'09)*, 2009, pp. 278–282.

[20] T. Godfrey *et al.*, "Modeling smart grid applications with co-simulation," in *Proc. 1st IEEE Int. Conf. Smart Grid Commun. (SmartGridComm)*, pp. 291–296, 2010.

[21] F. Ponci, A. Monti, and A. Benigni, "Simulation for the design of smart grid controls," in *Proc. IEEE 1st Int. Workshop Smart Grid Model. Simul. (SGMS)*, pp. 73–78, Oct. 2011.

[22] *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)–Framework and Rules*, Std 1516-2010 (Revision of IEEE Std 1516-2000), Aug. 18, 2010, pp. 1–38.

[23] R. Bottura *et al.*, "SITL and HLA co-simulation platforms: Tools for analysis of the integrated ICT and electric power system," in *Proc. EuroCon, Zagreb, Croatia*, pp. 918–925, Jul. 14, 2013.

[24] S. P. Carullo and C. O. Nwankpa, "Experimental validation of a model for an information-embedded power system," *IEEE Trans. Power Del.*, vol. 20, no. 3, pp. 1853–1863, Jul. 2005.

[25] D. Wang *et al.*, "Design of a novel wide-area backup protection system," in *Proc. IEEE/PES Transmiss. Distrib. Conf. Exhib.: Asia and Pacific*, 2005, pp. 1–6.

[26] K. Hopkinson *et al.*, "EPOCHS: A platform for agent-based electric power and communication simulation built from commercial off-the-shelf components," *IEEE Trans. Power Syst.*, vol. 21, no 2, pp. 548–558, May 2006.

[27] J. Nutaro, P. T. Kuruganti, L. Miller, S. Mullen, and M. Shankar, "Integrated hybrid-simulation of electric power and communications systems," in *Proc. IEEE Power Eng. Soc. Gen. Meet.*, Tampa, FL, USA, Jun. 24–28, 2007.

[28] H. Lin, S. S. Veda, S. S. Shukla, L. Mili, and J. Thorp, "GECO: Global event-driven co-simulation framework for interconnected power system and communication network," *IEEE Trans. Smart Grid*, vol. 3, no. 3, pp. 1444–1456, Sep. 2012.
[29] S. C. Muller, H. Georg, C. Rehtanz, and C. Wietfeld, "Hybrid simulation of power systems and ICT for real-time applications," in *Proc. 3rd IEEE PES Innov. Smart Grid Technol. Europe (ISGT Europe)*, 2012, pp. 1–7.
[30] J. C. Fuller, S. Ciraci, J. A. Daily, A. R. Fisher, and M. Hauer, "Communication simulations for power system applications," in *Proc. Workshop Model. Simul. Cyber-Phys. Energy Syst. (MSCPES)*, pp. 1–6, May 20, 2013.
[31] H. Georg, S. C. Muller, N. Dorsch, C. Rehtanz, and C. Wietfeld, "INSPIRE: Integrated co-simulation of power and ICT systems for real-time evaluation," in *Proc. IEEE Int. Conf. Smart Grid Commun. (SmartGridComm)*, pp. 576–581, Oct. 21–24, 2013.

Kyle Anderson (S'12) received the B.S. degree in electrical engineering in 2010, where he graduated as the Henry Ford Scholar; the M.S. degree in electrical engineering in 2012; and is currently pursuing the Ph.D degree from Stanford University, Stanford, CA, USA.

He has previously worked as a Software Engineer with Ericsson, Plano, TX, USA; Arista Networks, Santa Clara, CA; and Google, Mountain View, CA.

Mr. Kyle is a Mayfield Fellow, a Stanford Graduate Fellow (SGF), a recipient of the Frederick Emmons Terman Award, a recipient of the 2010 Agilent Award, and a member of Tau Beta Pi.

Jimmy Du received the B.S. degree in electrical engineering from Stanford University, Stanford, CA, USA. He is a Graduate Student at the same university where he is pursuing the M.S. degree in electrical engineering.

He is an Asian Pacific Islander American Scholarship Fund (APIASF) Scholar. His research interests include smart grid controls and distributed systems.

Mr. Du is an APIASF Scholar.

Amit Narayan received the B. Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 1993, and the Ph.D. degree in electrical engineering from the University of California at Berkeley, Berkeley, CA, USA, in 1998.

He was the Director of Smart Grid Research in Modeling and Simulation at Stanford University, Stanford, CA, USA, and currently is the Founder and CEO of AutoGrid Systems, Redwood Shores, CA. He has published over 25 papers in the area of design automation, holds seven U.S. patents, and is an Active Advisor to several startup companies in the bay area.

Abbas El-Gamal (M'71–F'12) received the B.Sc. (Hons.) degree in electrical engineering from Cairo University, Giza, Egypt, and the M.S. degree in statistics and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 1972, 1977, and 1978, respectively.

From 1978 to 1980, he was an Assistant Professor of Electrical Engineering with the University of Southern California, Los Angeles, CA. Since 1981, he has been a Member of Faculty with Stanford University, where he is currently the Hitachi America Professor with the School of Engineering. In 1984, he founded the LSI Logic Research Lab, San Jose, CA, which later became the Consumer Product Division. In 1986, he cofounded Actel, Mountain View, CA, where he served in several capacities, including Chief Scientist. In 1999, he cofounded Silicon Architects, where he was a Chief Technical Officer and a Member of the Board of Directors until Synopsys, Mountain View, acquired it in 1995. He was a Vice President of Synopsys from 1995 to 1997. He cofounded Pixim, Mountain View, in 1999 to commercialize the technology developed under the programmable digital camera program. He has also served on the board of directors and advisory boards of several other semiconductor, electronic design automation (EDA), and biotech startups. From 1997 to 2002, he served as the Principal Investigator on the Programmable Digital Camera Program, Stanford University. From 2004 to 2009, he was the Director of the Information Systems Laboratory, Stanford University. He was a Visiting Professor and MacKay Fellow with the University of California, Berkeley, CA, in Fall 2009–2010 and visited Tsinghua University, Beijing, China, as a Member of the Tsinghua Guest Chair Professor Group on Communications and Networking in Spring 2009–2010. His research interests include network information theory, field programmable gate array, and digital imaging devices and systems. He has authored or coauthored over 200 papers and holds 30 patents in these areas.

Prof. El Gamal is a Member of the National Academy of Engineering, Washington, DC, USA. He has received several honors and awards for his research contributions, including the 2012 Claude E. Shannon Award and the 2004 IEEE International Conference on Computer Communications (INFOCOM) Paper Award.