An Internet-Based Platform for Testing Generation Scheduling Auctions

Ray D. Zimmerman *rz10@cornell.edu*
Robert J. Thomas *rjt1@cornell.edu*
Deqiang Gan *deqiang@ee.cornell.edu*
Carlos Murillo-Sánchez *cem14@cornell.edu*

School of Electrical Engineering, Cornell University, Ithaca, NY 14853

Abstract

This paper describes the uses and architecture of a network-centered computing-rich software platform called PowerWeb. PowerWeb was designed and built as a simulation environment for experimentally testing various power exchange auction markets through tournaments. It is designed to host simulations of a competitive "day-ahead" electric energy market in the context of a restructured electric power industry. The PowerWeb environment is meant to be flexible so as to accommodate different "rules of the game". In this paper we describe its interactive, distributed and web-based character.

1 Introduction

The US electric power industry is taking major steps forward to restructure its institutional arrangements to support competition among energy suppliers. The US is not the first in the world to embark on this path and to refer to the undertaking as deregulation would be a mistake. In 1990 the United Kingdom restructured it's industry to form separate generation, transmission and distribution companies. Today, this arrangement represents perhaps the most complex regulatory environment in the world as a result of efforts to ensure that the independent companies provide reliable electric power at fair prices. Despite the experience in the UK, the historical experience with deregulation of other industries has been an unqualified success from the point of view of economic efficiency. For example, price decreases in the airline, natural gas, and long distance telephone industries have been well documented [1]. However, the electric utility industry presents unprecedented complications for restructuring. In particular, electric power networks offer multiple simultaneous commodities and a variety of externalities such as reliability concerns that imply a pure market solution is unlikely to be efficient. The unbundling of technical services suggests the existence of a multi-dimensional or multi-unit market where the sale of many related goods will take place. While there is an emphasis by economists on efficiency, there is little known about the efficiency properties of various auction designs for multiple objects.

The move to competitive markets for electric power is advancing at increasing speed, based on the notion that competition will generate cost savings. In our opinion there is insufficient attention being paid to the type of market to be employed. The notion that any market is better than no market is demonstrably false for a number of reasons. Without careful attention to the design of these markets the promise of deregulation could easily be lost through new types of inefficiencies. For example, it has been shown in experimental economics that the specific auction institution (double auction, call auction, uniform price auction, English auction, etc.) can have dramatically different efficiencies. Some auctions are much more efficient in the face of market power than others [2]. Efficiency differences as much as 15% are commonly observed.

Although it has been shown by Smith [3] in economics laboratory experiments that reasonable efficiencies can be achieved in smart markets for extremely simple network situations, no experiments have been conducted testing smart markets with complex networks. Additionally, no experiments have tested multiple interconnected markets for ancillary services along with the energy market as have been proposed for most electric power markets. The unit commitment problem remains untouched in experimental testing except for the primitive yet intriguing experiments of Plott [4]. The common thread in all of these untested areas is the necessity for collaborative research in electrical engineering and experimental economics. It is for this reason that we have designed and built the experimentation environment we refer to as PowerWeb. In the remaining part of this paper we describe its architecture in some detail. It is an example of a network-centered computing environment that we believe will become commonplace in the years to come.

2 PowerWeb functionality

PowerWeb is an Internet-based simulation environment for testing various power exchange auction markets experimentally using human decision makers. It is interactive, distributed and web-based. It is designed to host simulations of a competitive "day-ahead" electric energy market in the context of a restructured electric power industry.

1060-3425/98 $10.00 (C) 1998 IEEE

2.1 Overview

Since PowerWeb is based on the Internet, it is not necessary for participants to be in the same physical location in order to conduct an experiment. The web-based architecture, shown in Figure 3, enables a participant to access PowerWeb from anywhere Internet access is available. The only software necessary is a modern web browser, such as Netscape Navigator™, which runs on nearly all computing platforms in common use today.

The PowerWeb environment is meant to be flexible so as to accommodate markets with a variety of "rules of the game". Because of operational constraints on a power system, it seems necessary to have a central agent acting as an independent system operator (ISO). PowerWeb is designed to host various ISO models, for example, a "maximum ISO" where full market information is available. This is typical of several variants of the PoolCo model. The PowerWeb environment is designed to run unit commitment and optimal power flow routines against load forecasts in order to provide generation schedules such as those that might be assigned by a Power Exchange (PX).

In the current implementation of PowerWeb, the ISO/PX receives offers to sell power from independently owned generation facilities. Based on a forecasted load profile for the next day and the information gathered from the generator's offers, the ISO computes the optimal generator set points along with a corresponding price schedule which will allow the system to meet changing demand while satisfying all operational constraints. The method used to solicit offers and the mechanism which determines prices are dependent on the market model being examined.

As a web-based tool, PowerWeb may be used in several capacities. It can be utilized in a tightly controlled setting where a well-defined group of subjects are used for a very specific set of market experiments. It can also be used in a more open environment in which anyone on the web can log in and "play" as a generator competing against other generators, controlled by other humans or computer algorithms (automatons), to generate power profitably.

In either case, since PowerWeb is web-based it is accessible at all times to anyone with proper authorization, as long as the servers are up and running. To eliminate the need to coordinate accesses (via phone, e-mail, etc.) and to prevent one user's actions from interfering with another's, all accesses occur in the context of a given "session".

2.2 A typical session

When initially accessing PowerWeb, it is necessary to register to obtain a user id and password which will be used to authorize all further access. A registered user can log in to an existing session, and eventually they will be able to create a new session via a set of HTML forms. The session specifies which power system is being simulated, who "owns" which system resources (generators, etc.), and what market mechanism is in use. Multiple sessions can be active at any given time and activity in each is completely independent of the others. Typically, a user in a session will "own" one or more generating plants, or may represent the ISO. The ownership mapping may be static, or it may be set to update dynamically as participants enter and leave the session.

After logging in, a user has access to the system information area which gives tabular summaries of the system operation conditions as well as a "live" one-line diagram of the power system. Figure 1 shows the one-line diagram of a 6 generator, 30 bus system in PowerWeb's database. This diagram is generated dynamically by a Java applet from information retrieved from a relational database server. The diagram can be panned and zoomed and it is interactive in that clicking on an object such as a line, bus, generator, or load will query the database for information about the object. For example, selecting a bus will display the current information about real and reactive flows into and out of the bus as well as information about the current voltage level of the bus. This information is the most recent power flow data based on the current unit commitment and dispatch schedule. Access to the information is granted depending on the identity of the one requesting it. For instance, access to a competing generator's cost information would not be permitted.

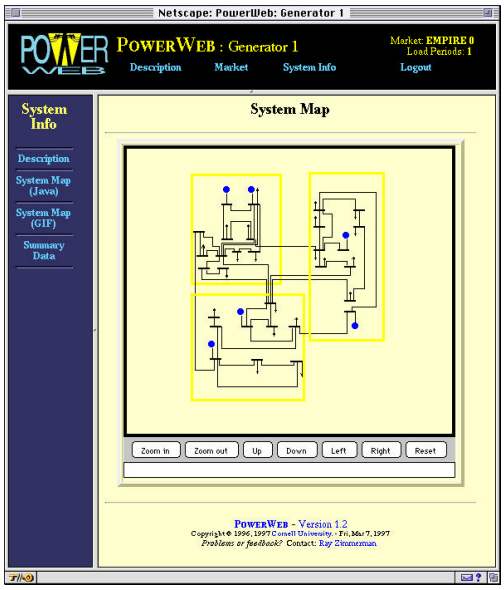

Figure 1: PowerWeb one-line diagram display, showing 30-bus system

Market information is also available to the user, including the cost, offer, dispatch and revenue information for each period for any generators owned by the

user. The main auction page, shown in Figure 2 for a simple sealed bid type auction, allows for submission of offers into the auction. The plot which displays the costs and the block offers is drawn by a Java applet, updating automatically as offers are entered and revised.

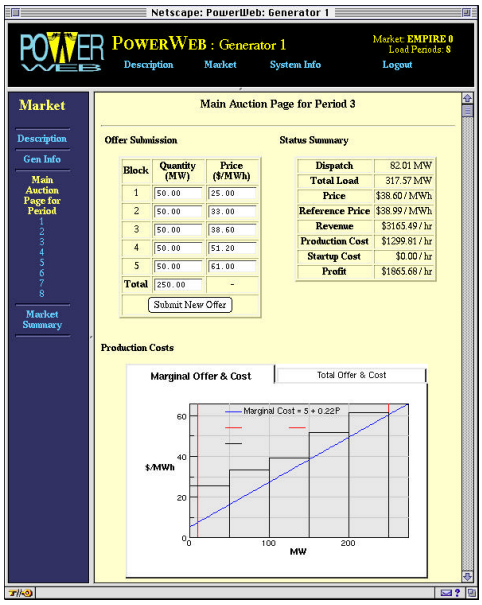

Figure 2: The form for entering bids. The "blocks" are displayed dynamically via a Java applet

Though not implemented in the current version, PowerWeb is also designed to display auction results after an experiment so the experimenters and participants can visually observe key aspects of the behavior of the market.

When a session has ended or a user has finished they can log out explicitly, or quit their browser which implicitly does an automatic log out.

The PowerWeb User's Manual [5] has more details regarding PowerWeb's functionality.

3 Internet technologies

In order to understand some of the design choices that were made for PowerWeb, it is important to understand the capabilities and limitations of the currently available Internet technologies. These technologies include a rich collection of cross-platform, open standards that enable developers to quickly create and deploy network-centered applications.

This section explores some of the primary technologies utilized in PowerWeb. It should be noted that, for many of these technologies, the Internet is not the only, or necessarily even primary, context for their use. PowerWeb is a distributed application defined by various *programs* running simultaneously on different computers and the *protocols* by which these programs interact. PowerWeb uses a client-server architecture, where the programs involved take on the role of client or server for a specific of interaction. The technologies discussed below are divided into the *languages* used to implement PowerWeb's various programs and the *protocols* by which they communicate.

3.1 Languages

HTML [6]

HyperText Markup Language (HTML) is a very well-known and widely used international standard maintained by the Internet Engineering Task Force (IETF) for defining a document with possible links to other network resources. An HTML document, as interpreted and rendered by a typical web browser, may include structured and formatted text, tables, fill-out forms, images, hypertext links, Java applets and references to other types of data which can be handled via helper applications or browser plug-ins.

HTML is ideal for displaying information to a user on the web. An HTML renderer is built into every web browser so it is very cross-platform in nature. It is limited in that it is static, so interactivity with an HTML document is generally in the form of a link to another (possibly dynamically generated) HTML document. The vast majority of the user interface in PowerWeb consists of dynamically generated HTML pages.

Perl [7]

Perl is a language originally designed as a UNIX administration tool. It has become tremendously popular with web developers as a language for writing programs which generate HTML pages as output. One of Perl's many strengths is in the area of text handling, which is exactly what is needed for producing and manipulating HTML. Perl's operating system, file system, network and database interface capabilities along with its object-oriented language features, uniquely coordinated developer community, and extensive archive of high quality freely available reusable modules, make it an ideal choice for many of PowerWeb's tasks

Java[8]

Java is a complete programming language that allows true platform-independent application development. It was developed by Sun Microsystems and has been submitted to the open standards process. It is an object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multi-threaded, and dynamic language. Of particular significance to developers of network applications is the ability of a Java application class, called an "applet", to be securely downloaded from anywhere on

the network. The application can then be loaded dynamically and executed immediately. It is simple to place references to Java applets into an HTML document. Users may then invoke an applet by simply accessing the relevant page.

PowerWeb currently uses Java applets to display the one-line diagram of the power system and to display cost and offer information graphically. In spite of the fact that Java is a relatively immature technology and there is some inconsistency across implementations, it promises to be a dominant player in the development of network-centered applications like PowerWeb.

JavaScript [9]

JavaScript is an interpreted scripting language developed by Netscape Communications. Contrary to the implication of the name, it is not based on Java. JavaScript code can be embedded within an HTML document where it is executed by the web browser in response to specified events. For example, a button can be linked to some JavaScript code that executes when the button is pressed.

One area where JavaScript is used in PowerWeb is to validate offers which have been entered in a form before sending them back to the server.

SQL

The Structured Query Language (SQL) is a standard language for defining, querying and manipulating the data in a relational database. PowerWeb uses the SQL to access and modify all data in its database.

Matlab

Matlab, the language, is an interpreted, procedural language developed by The MathWorks and designed for numerical mathematics, especially applications involving matrix and vector computations. It includes highly optimized dense and sparse matrix factoring routines among many others. Until the most recent version, Matlab was quite limited in the data structures available, but its strength in matrix and vector computations still make it a tool of choice for the types of computations required for power system simulations.

PowerWeb uses Matlab as the language for implementing all of the optimal power flow programs as well as the market pricing code which form the core of PowerWeb's computational server. Since Matlab is not explicitly designed as a network language, it was necessary to develop our own protocol for interacting with the Matlab programs.

3.2 Protocols

HTTP [10]

The HyperText Transfer Protocol (HTTP) is the standard protocol for communicating between clients and servers on the web. HTTP is a stateless protocol which specifies how a client and server establish a connection, how the client requests a specific service from the server, how the server issues a response, and how the connection is terminated. The terms "client" and "server" are defined primarily in terms of their roles in an HTTP interaction. HTTP connections over the Internet are implemented using the TCP/IP protocol.

In PowerWeb all interaction between the web browser and the web server are based on HTTP, as are all communications with the computational server.

CGI [11]

Common Gateway Interface (CGI) is a very popular standard protocol for communication between a web server and an external program, typically referred to as a CGI program. The primary role of a CGI program is to dynamically create data on demand, such as a web page or image, for the web server to return to a client. Since the CGI protocol clearly defines the interface to the web server, any language which can implement this interface can be used to write a CGI program. One of the limitations of CGI is the performance penalty arising from the overhead involved in spawning a new process for each request. FastCGI [12] is a lesser used alternative which allows the external program to continue to run between requests to avoid this overhead. Some web servers also have application programming interfaces (APIs) which allow developers to directly extend the web server functionality to be able to generate dynamic pages.

PowerWeb uses CGI programs implemented in Perl for nearly all of the dynamically generated HTML pages which make up PowerWeb's user interface. Some of the other alternatives mentioned are also being considered.

Cookies [13]

An HTTP cookie is an object containing state information, a simple name and value pair, that a web server informs a web browser to send along with any subsequent requests to a specified range of URLs. PowerWeb utilizes cookies to store login information to avoid requiring a user to type in their password for each page they want to access.

URL [14]

A Uniform Resource Locator (URL) is the standard means of identifying and locating any network resource. URLs are used to address specific web pages

including those which may be generated dynamically by a program. PowerWeb uses URLs to identify the HTML documents that make up its user interface.

MIME [15]

The Multipurpose Internet Mail Extensions (MIME) standard specifies the type of data and encoding associated with a document. PowerWeb uses MIME to specify the type of data transmitted over the HTTP connections between clients and the web server and computational server.

DBI [16]

DBI is database interface module and API for accessing SQL databases from Perl. It is designed to be independent of the database server being used. It is an ideal interface for PowerWeb to use to access its database server from the Perl CGI programs.

JDBC [17]

The Java Database Connectivity (JDBC) is the standard protocol for accessing a database from the Java programming language. Any additional Java-based PowerWeb components that are developed will use JDBC for all database access.

4 Communications architecture

PowerWeb employs a distributed architecture on several different levels. First, it is a client server architecture, in that all user interaction with PowerWeb is via a web client (a browser, or applet running within a browser) communicating with the PowerWeb server. Second, the PowerWeb server also has a distributed architecture consisting of several independent processes, such as the web server, the database server, and the computational server, each of which can be running on different computers. Even the computational server has several parts which need not reside on a single machine.

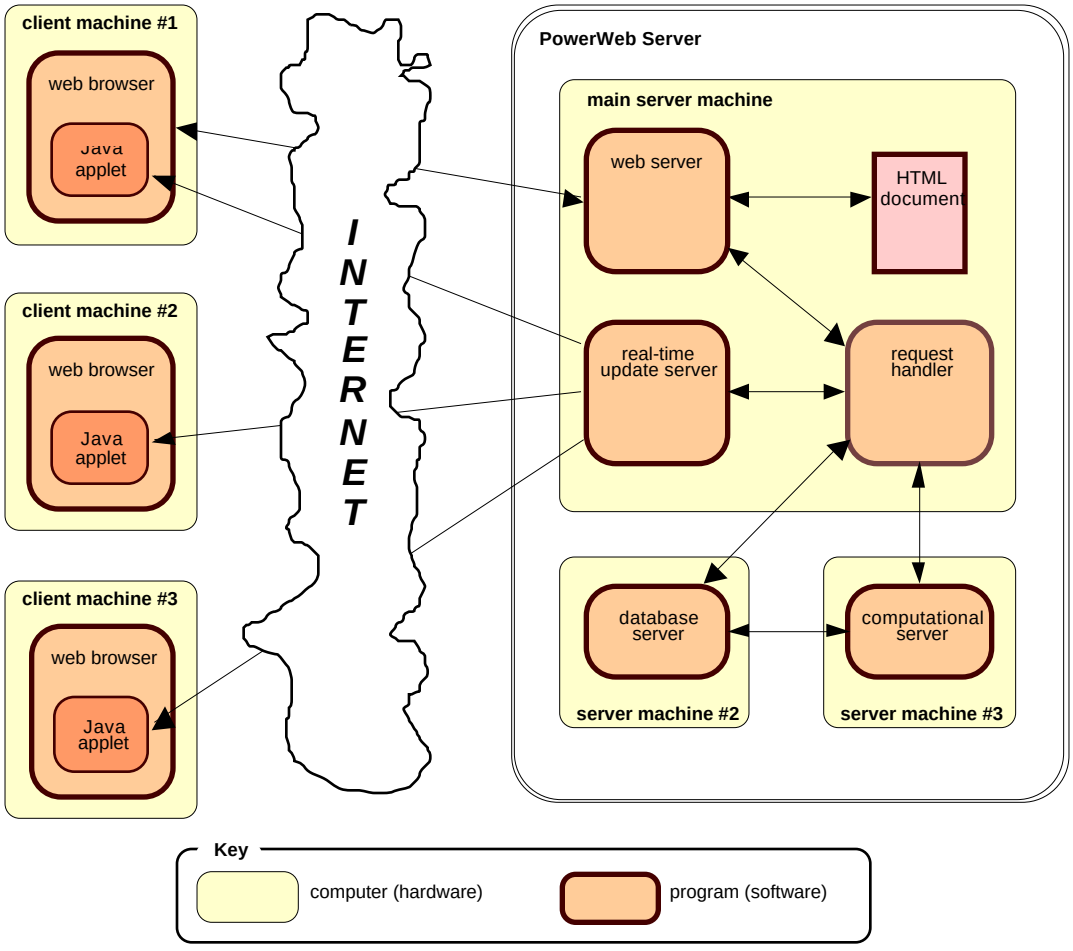

Figure 3 PowerWeb communications architecture

Figure 3 illustrates the various components of the PowerWeb communications architecture. Currently, the entire PowerWeb server is running on a Sun Ultra 2200, with dual 200 MHz processors. The web server is an Apache server [18] and the database server is mySQL [19].

When a request for a specific URL is transmitted from one of the web clients to the web server, the server determines whether the URL refers to a static HTML document or to one which must be generated on the fly. In the first case, the web server retrieves the file from the disk and returns it to the client. In the second case, the web server passes the request on to what is referred to here generically as a "request handler". This could be a separate program invoked via a CGI or FastCGI protocol, or it could simply be a server module which runs to produce the document to be returned.

The vast majority of accesses to PowerWeb are processed by a "request handler" which, in turn, makes requests to the database server. In the current implementation, the "request handler" is a CGI program written in Perl. All accesses subsequent to login are accompanied by a cookie containing authentication data which is compared with information provided by the database. The protocol for communication with the database server is a specialized protocol defined by the developers of mySQL which uses UNIX sockets for local communication and TCP/IP sockets for remote communication. The computational server also receives requests from the "request handler" via the HTTP protocol. Based on the parameters sent with the request, it retrieves the necessary data from the database, runs the requested computation, and returns the output to the "request handler". The computational engine is implemented as a web server with a CGI program that makes database queries and invokes Matlab programs through a UNIX pipe.

The real-time update server, which is not currently implemented, is needed in order to contact a client when it needs to be made aware of new information. The standard web protocols facilitate only the following sequence: a client connects to the server with a request, the server responds and closes connection. The HTTP protocol is a state-less protocol which does not provide the ability for the server to initiate a communication with a client. One way to overcome this limitation is for the server to keep a "live" connection to a Java applet running at each active client. This "live" connection must be handled via another server process. A typical use of the update server would be if the user acting as ISO triggers a recomputation of the dispatch and price schedules, when the computation is completed, the server would pass a message to the update server to notify the clients to retrieve the new information.

5 Database structure

In the PowerWeb environment there is a tremendous amount of data which needs to be handled and used in varying contexts. A relational database server satisfies the needs for logical data organization with its relational model, synchronized updating of the data to maintain data integrity, and flexibility in access to the data via the SQL language.

In the interactive Internet-based environment, performance of the database server is also of utmost importance. The mySQL server [19] used in PowerWeb meets these requirements nicely.

The data handled by PowerWeb can be classified into three main categories:

- user administration data
- power system data
- session data

5.1 User administration data

The user administration data is used primarily to control who has access to what information in PowerWeb. When a request is made for user *X* to see cost information for a generator "owned" by group *Y*, for instance, these tables would be accessed to determine whether the request is coming from someone who is authenticated as user *X*, and to ensure that user *X* really is a member of group *Y*. These data are stored primarily in three tables:

Users	user id, password, registration info
Groups	group id and name
UserGroup	mapping of users to groups

5.2 Power system data

The power system data refers to the coordinate data needed to display a one-line diagram of the system, all of the power flow data needed to run an optimal power flow (OPF), the results of the OPF, and the cost information required to compute profits given the resulting dispatch and price schedules. These data are held in the tables described below.

The main table which contains the top level data for each base case, one row per case, is the *Systems* table. Each row of each of the *Areas*, *Buses*, *Branches*, *BranchSegments*, *Caps*, *Gens*, and *Loads* tables has a field which links it to a system in the *Systems* table and another which is the index of that particular area, bus, branch, etc. within that system. In addition, each row of the tables contain the following data:

Systems	system id, system name, MVA base, number of buses, lines, gens, etc.
Areas	area name, price reference bus

Buses area number, zone, voltage class, bus type, upper and lower voltage limits, initial voltage magnitude and angle, one-line coordinates

Branches "from" and "to" bus numbers, circuit number, line status, impedance and charging parameters, thermal capacity ratings, tap ratio and phase angle shift

BranchSegments one-line coordinates of branch segments

Caps bus number, status, admittance, one-line coordinates

Gens bus number, gen name, baseMVA, status, initial active and reactive power generation, initial voltage magnitude set-point, upper and lower limits on active and reactive output, ramp rate, min up and down times, key to entry in *GenCosts* table

GenCosts cost class, cost model, startup and shutdown costs

GenCostData key to entry in *GenCosts* table, parameters which define polynomial or piece-wise linear production cost curve

Loads bus number, real and reactive demand, percentages of constant impedance, constant current, and constant power

The data for the (optimal) power flow solutions is stored in separate tables, to avoid having to store all of the constant power flow data for each case that is run. The *Solns* table holds the top-level data, one row per solved case, to which the other tables are referenced. Each of the *BusSolns*, *BranchSolns*, and *GenSolns* tables has a column which links each record to an entry in the *Solns* table and another which is the bus, branch, or generator index. In addition, the following information is stored in theses tables:

Solns solution id, name, system id

BusSoln bus type, voltage magnitude and angle, Lagrange multipliers for real and reactive power balance requirements, Kuhn-Tucker multipliers for upper and lower voltage constraints

BranchSoln real and reactive power flow at each end of the branch, Kuhn-Tucker multipliers for flow constraints

GenSoln status, active and reactive power output, voltage setpoint, Kuhn-Tucker multipliers for upper and lower real and reactive output constraints

The *Changes* and *ChangeData* tables provide a convenient way to specify modifications to an existing base case. Each row in *Changes* corresponds to an independent set of modifications that can be applied to an existing base case. *ChangeData* has a column which associates that particular change with a corresponding set in *Changes*.

Changes change set id, valid system id (0 for any system), name of change set

ChangeData table, column and index of data to be modified (all indices if index is 0), type of change (scale or replace), scale or replacement value

5.3 Session data

All interaction with PowerWeb is in the context of a "session" that the user is logged in to. The tables listed in this section handle the data for managing these sessions. The main top-level data is stored in the *Sessions* table, with one row per session. Many of the other tables here have references to a particular entry in this table which link their data to a specific session.

Sessions session id, name, user id of session owner, system id, market id, number of trading periods, number of dispatch periods, number of iterations per trading period, length of dispatch period, time given for each iteration, persistence level, synchronous vs. asynchronous ISO offer evaluation, logging detail level, creation time, start time, simulation clock time, session state, current trading period, current dispatch period, current iteration number, textual session description

The *Markets* table has an entry for each type of market implemented in PowerWeb. The table stores a few parameters common to the various markets, but the rules of each market are programmed separately in PowerWeb code.

Markets market id, name, type of offer (blocks, functions), auction id, offer dimensions

The *Resources* and *ResOwners* tables specify which system resources (e.g. generators) are "owned" by which user or group.

Resources resource id, session id, resource type (gen, load), index, name

ResOwners resource id, user id or group id

Each PowerWeb session is organized into a sequence of trading periods, during which offers are made to sell power to meet a forecasted demand schedule. The schedule for a given trading period may be divided into several dispatch periods, each of which has its own demand forecast, its own actual demand, and possible

it's own set of other arbitrary changes to network parameters. A "system profile" is used to specify how the system parameters, including demand, vary through out the various trading periods and dispatch periods. The *SystemProfiles* table defines the system parameters used for each period.

SystemProfiles session id, trading period, dispatch period, type of profile (forecasted or actual), sequence number, change set id

In addition, the submission of offers and computation of dispatch and price schedules may be iterated several times for each period in the system profile. Each of these iterations is treated as a separate "case" to be run. The *Cases* table associates an id with each iteration of each period which the other tables can use as a reference. *CaseIOData* associates each case with it's set of offers and dispatch results, and *CaseSolnData* matches each case with its solution in the *Solns* table.

Cases case id, session id, trading period, dispatch period, iteration, locked/unlocked

CaseIOData case id, resource id, type of IO data (real power, reactive power, etc.), offer id, dispatch id

Offers offer id, sequence (block) number, quantity, price

Dispatches dispatch id, quantity, price, fixed cost, variable cost, startup cost, penalty, profit

CaseSolnData case id, solution id

6 Underlying optimal power flow

At the heart of the PowerWeb computational engine is an optimal power flow (OPF) program that is executed by the ISO in response to offers submitted in an auction. The market activity rules determine what offers are valid, but it is the ISO's role to ensure the safe and reliable operation of the network. By using an OPF, the ISO can legitimately allocate generation in an "optimal" way while respecting line flow constraints, voltage magnitude constraints, VAr constraints and any other constraints that are necessary to ensure safety and reliability. As a by-product, the OPF also produces the shadow prices associated with locationally-based marginal pricing (LBMP) of power. These prices can be used as determined by the market mechanism being employed.

In the context of a market in PowerWeb, the OPF may be subjected to widely varying costs and therefore dispatches which are far from typical base case operation. It is important in such an environment that the OPF be extremely robust.

The OPF in PowerWeb can handle quadratic cost functions as well as convex piece-wise linear cost functions. The OPF problem can be stated as follows:

Minimize the total cost of generation,

$$\min_{\mathbf{P}_g, \mathbf{Q}_g} \sum f_i$$

such that active and reactive power balance equations are satisfied,

$$P(V,\theta) - P_{gi} + P_{Li} = 0$$

$$Q(V,\theta) - Q_{gi} + Q_{Li} = 0$$

active and reactive generator outputs are within specified limits,

$$P_{gi}^m \le P_{gi} \le P_{gi}^M$$

$$Q_{gi}^m \le Q_{gi} \le Q_{gi}^M$$

bus voltages lie in an acceptable range,

$$V^m \le V_i \le V^M$$

and apparent power flow at the "from" and "to" bus ends of each line do not exceed the line's capacity limits,

$$\tilde{S}_{ij}^f \le S_{ij}^M$$

$$\tilde{S}_{ij}^t \le S_{ij}^M .$$

In addition to the formulation above, PowerWeb auctions require the ability for the OPF to de-commit units with excessive offers.

Several OPF algorithms have been developed in Matlab for use in PowerWeb. One method is based on the "constr" function in Matlab's Optimization Toolbox (OT). This optimizer uses a quasi-Newton approach which seems to work quite well for small systems. Quadratic cost functions can easily be handled directly. The piece-wise linear costs arising from block offers in PowerWeb are accommodated by introducing cost variables which are constrained below by the piece-wise linear functions.

The other OPF algorithms available to PowerWeb are based on successive linear programming. LP-based methods have been examined extensively and are used in some production grade OPF packages [20]. Three LP-based OPF solvers have been developed for PowerWeb. One is a "dense" approach which eliminates the power flow equations and network voltage variables in the formulation of the LP. The dense method also uses one variable for each segment of a piece-wise linear cost function [21]. The other two approaches solve a larger sparse LP which contains the power flow equations and bus voltages. One uses the "one variable per cost segment" approach used in the dense method and the other uses the constrained cost variable approach employed by the OT-based OPF to handle the piece-wise linear cost functions.

Since the OT's quasi-Newton based "constr" function does not preserve sparsity it is only suitable for small problems. With a good, sparse LP-solver the

LP-based methods are able to handle much larger systems. The following are some preliminary timing results in seconds for the first two LP-based methods running on a Sun Ultra 2200, using *BPMPD*, by Csaba Mészáros, as the LP-solver [22].

Table 1: OPF Timing Results (seconds)

Size of Test System		Run Time (seconds)	
Generators	Buses	Dense LP	Sparse LP
3	9	2	3
6	30	8	12
7	57	13	13
19	118	30	27
54	118	65	43
80	148	408	675
64	300	150	150

In the current implementation, the ability to decommit expensive generators is handled by a commitment heuristic. However, if an expensive generator is needed for VArs, this is only detected if the OPF fails to converge. Next stage developments for the OPF are to improve the handling of unit commitment [23] and to include reliability and dynamic security constraints.

The latest version of the Matlab OPF solvers and more detailed documentation of the algorithms employed are available at no cost at http://www.pserc.cornell.edu/matpower/ as part of the MATPOWER package [24].

7 Conclusions

In light of the restructuring of the electric power industry to foster a competitive environment among energy suppliers, tools for experimentally testing the various proposed market structures are needed. PowerWeb is designed to be a flexible platform for performing such economic experiments using realistic modeling of the physical network and real human decision makers. As an Internet-based, network-centered computing environment, PowerWeb makes use of a wide variety of technologies in its implementation. All data are handled by a relational database and market computations are performed by a Matlab-based OPF. Next stage enhancements to the OPF involve incorporating security constraints. On-going development of the PowerWeb platform is planned to accommodate the industry's experimental needs.

8 References

1. C. Winston, 1993. "Economic Deregulations = Days of Reckoning for Microeconomists" *Journal of Economic Literature*, vol. 31: 1263-1289.
2. J. Bernard, W. Schulze, and T. Mount, 1997. "Auction Mechanisms for a Competitive Electric Power Market", paper presented at the American Agricultural Economics Association Summer Meetings, July 1997, Toronto, Can.
3. V. Smith, 1996, "Market Power and Mechanism Design for Deregulated Electricity Networks" Selected Paper presented at the Economic Science Meetings, Oct. 17-20, 1996, University of Arizona. Tucson, Arizona.
4. C. Plott, March 10, 1997, "An Experimental Test of the California Electricity Market", http://www.energyonline.com/wepex/reports/reports2.html.
5. R. J. Thomas, R. D. Zimmerman, R. Ethier, "PowerWeb User's Manual", PSERC 97-10, http://www.pserc.wisc.edu/pubs/PWebMan.pdf.
6. "IETF - HyperText Markup Language (HTML) Working Group", http://www.ics.uci.edu/pub/ietf/html/.
7. "The http://www.perl.com/ Home Page", http://www.perl.com/.
8. "Java Documentation", http://www.javasoft.com/docs/.
9. "JavaScript Guide", http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/.
10. "IETF - Hypertext Transfer Protocol (HTTP) Working Group", http://www.ics.uci.edu/pub/ietf/http/.
11. "The Common Gateway Interface", http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.
12. "FastCGI", http://www.fastcgi.com/.
13. "Persistent Client State HTTP Cookies" http://www.netscape.com/newsref/std/cookie_spec.html.
14. "IETF - Uniform Resource Identifiers (URI) Working Group", http://www.ics.uci.edu/pub/ietf/uri/.
15. "MIME (Multipurpose Internet Mail Extensions", http://www.oac.uci.edu/indiv/ehood/MIME/MIME.html.
16. "DBI - A Database Interface Module for perl5", http://www.hermetica.com/technologia/DBI/.
17. "The JDBC™ Database Access API", http://www.javasoft.com/products/jdbc/.
18. "Apache HTTP Server Project", http://www.apache.org/.
19. "mySQL Home Page", http://www.tcx.se/.
20. O. Alsac, J. Bright, M. Prais, B. Stott, "Further Developments in LP-based Optimal Power Flow", *IEEE Trans. On Power Systems*, vol. 5, no. 3, 1990, pp. 697-711.
21. B. Stott, J. L. Marino, O. Alsac, "Review of Linear Programming Applied to Power System Rescheduling", 1979 PICA, pp. 142-154.
22. C. Mészáros, "The efficient implementation of interior point methods for linear programming and their applications", Ph.D. Thesis, Eötvös Loránd University of Sciences, 1996.
23. C. Murillo-Sánchez and R. J. Thomas, "Thermal Unit Commitment Including Optimal AC Power Flow Constraints", a paper presented at and published in the proceedings of the 31st HICSS Conference, Kohala Coast, Hawaii, Jan 6-9, 1998.
24. R. Zimmerman and D. Gan, "MATPOWER: A Matlab Power System Simulation Package", http://www.pserc.cornell.edu/matpower/.