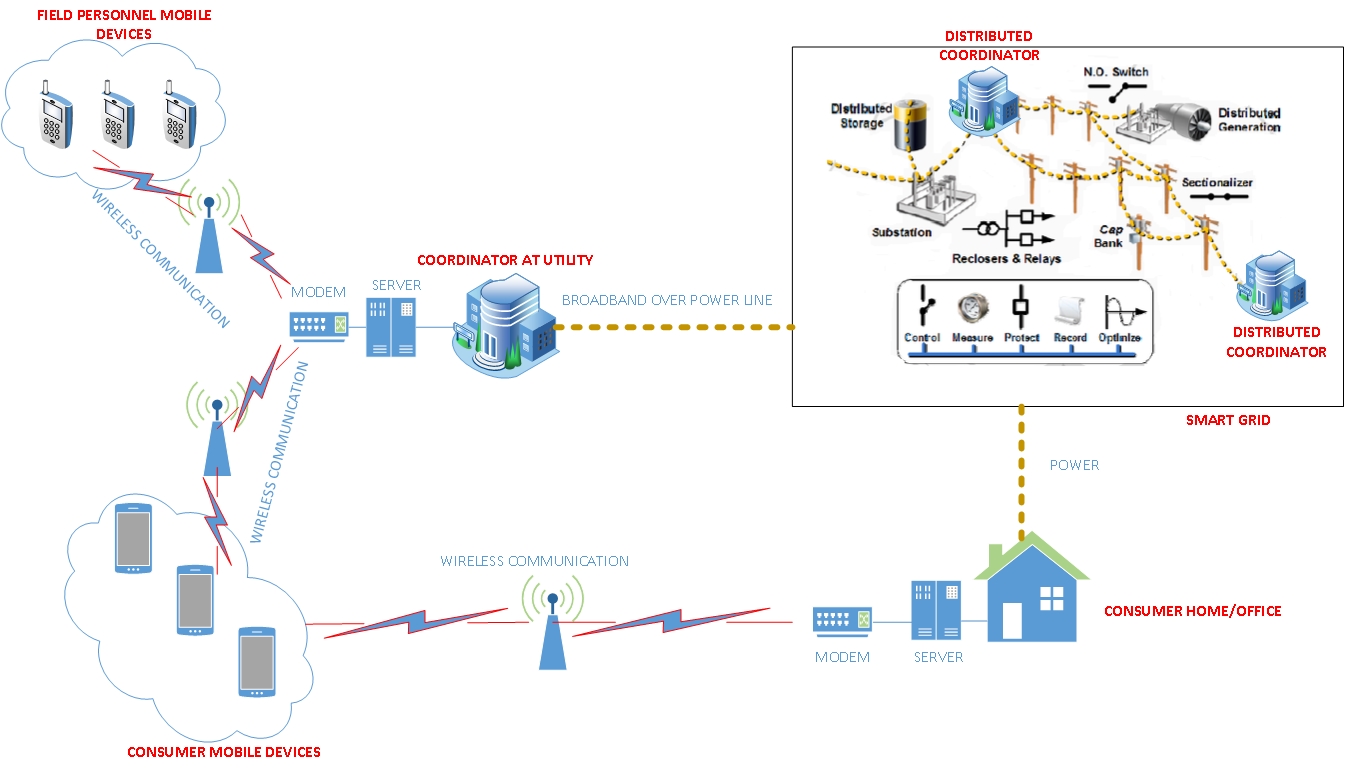
# Final Project

## Mobile Devices and Big Data (Fall 2015)

### 1.1 System Architecture

*Figure* 1 shows the system level architecture of the platform. The servers and modems are part of the Consumer homes/offices and of the Coordinators.



**Figure 1. System Architecture**

Mobile devices with the Consumers will be used in the following ways (Consumer has control over all data and schedule privacies):

1. For transferring detailed (i) appliance models and (ii) renewable power source models and their dynamic capacities at the Consumer home/office, from the Consumer home/office server to the utility server.
2. For reading the variable pricing model from the utility for optimizing appliance schedules.
3. For scheduling appliances in the user homes/offices and notifying the utility about the schedule. This scheduling can either be done entirely on the Consumer home/office server, or entirely on the Consumer mobile device. However, ***scheduling can also be dynamically partitioned*** between the Consumer servers and their mobiles devices (which are often powerful computing devices with 4 or more cores) to improve response time and user privacy. Such partitioning will depend on server availability, on mobile device computing capability and battery life, and on network bandwidth.
4. Since the utility communicates with all Consumers, it has a global view of all schedules in its distribution network, and hence, it can globally optimize for all Consumers in its server. Consumer specific schedules are thereafter sent back to the respective mobile devices for the Consumers to accept or override.
5. Since Consumers can also assume the roles of Suppliers, they can trade renewable power in the market provided by the utility company based on their time varying productions and demands.

Mobile devices with utility field personnel will be mainly used for grid monitoring and reading task assignments from the utility server for fast response to outages.

The utility server (Utility Coordinator in *Figure* 1) will: (i) store detailed grid and appliance models of the Consumer homes/offices, (ii) get real-time and projected power demand, capacity and price updates from the Consumers, Suppliers and the trading market, (iii) run optimizations at the system level for all Consumer schedules and notify the Consumer mobile devices, (iv) store model libraries and databases corresponding to the smart grid and the user homes/offices, and run mixed simulations and hardware-in-the-loop emulations (explained in section 3.3), and (v) monitor field data in the smart grid about power quality and grid resource status, and notify field personnel mobile devices. We believe that distribution of such ***coordinators*** in different hierarchies of the smart grid will improve their efficiencies with respect to controllability, observability and real-time response to grid and user dynamics.

### 1.2 Schedule Optimization

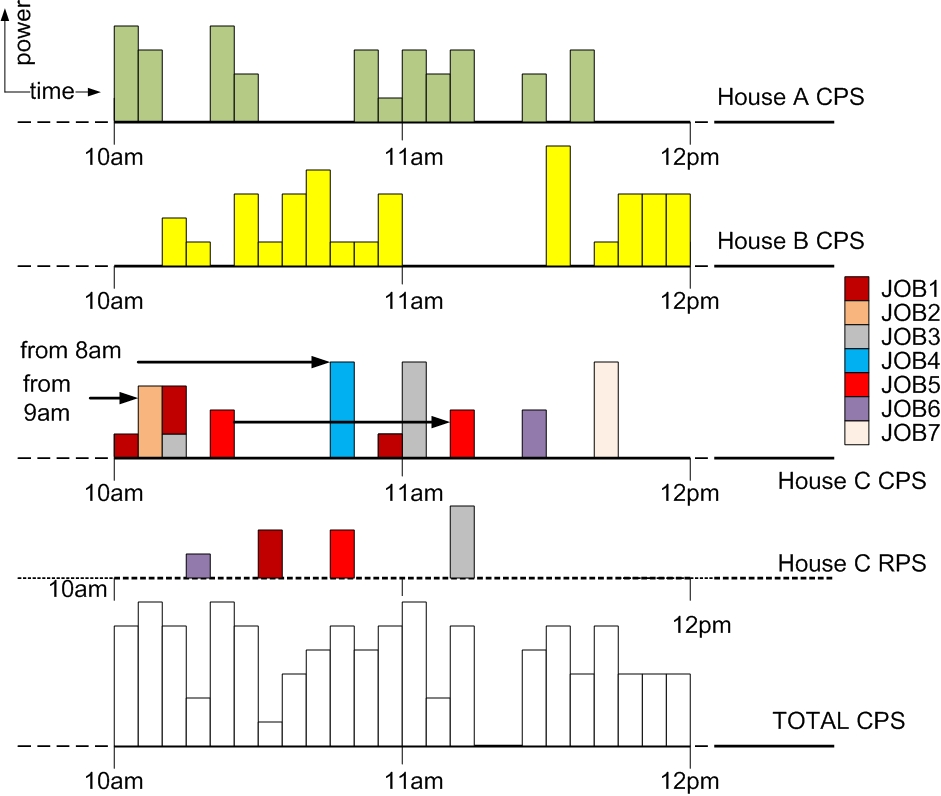
We shall now discuss the requirements of the Demand Side Management (DSM) optimization framework – this project. For ease of discussion, we consider a community of three houses (A, B and C) on the same commercial feeder, each one with its own exclusive renewable power source (RPS). The idea is to optimize the load scheduling of all appliances in all the houses to reduce the peak power demand on the commercial power supply (CPS) at any instance, while simultaneously reducing the energy demand on the CPS by shifting loads onto the RPS of the houses (subject to generation and storage capacities of their respective RPS). Hence, we can achieve reduction of total cost of electricity (T$) to the consumer.

To achieve this goal, use a distributed optimization scheme (DOS), with all houses in the community have their own copies of an optimization algorithm (OPT) running on their respective servers and/or mobile devices - the goal is to optimize the CPS load scheduling, not only for the particular house, but also to reduce the peak CPS demand over all houses. Once OPT at a house is executed on its server and/or its mobile device, the CPS schedule of that house is sent via the mobile device to the utility server as a *COMMIT*. The utility server broadcasts the current overall CPS scheduled loading to all house servers and their mobile devices. The entire DOS cycle is repeated every 5 minutes to enable the utility to track the peak demand on the CPS, while the real-time price of power is updated every 1 hour.

*Figure* 2 shows a snapshot of the DOS when OPT is about to be executed in house C. Only the loads on the CPS of houses A and B are relevant and shown in the figure, while both the CPS and RPS loads are shown for house C. Using scheduling terminology, we refer to the assignment of a load or appliance to a certain time instance, as the scheduling of the corresponding job. Any such job is tied to a particular set of appliances, all of which have the same time range of execution bounded by a start-time and a deadline. The total power of the job is known by adding the power demands of the individual appliances. The energy requirement of an appliance is the product of its power and required run time. The total energy requirement of the job is found by adding the individual energy requirements of its mapped appliances. In the degenerate case, only one appliance can be mapped to such a job. We depict time along the x-axis (in 5 minute intervals) and power of the different jobs along the y-axis. Individual jobs have been separately shown for house C because only these jobs can be rescheduled in this OPT step; jobs at other houses have been combined. The height of a vertical bar corresponding to a job at any time instant *t*, corresponds to its power demand at *t* because of appliances (mapped to that job) which are active at *t*. The total energy consumption of an appliance with color *c* can be found by adding the heights of all bars of color *c* over an entire 24 hour period of optimization. In *Figure* 2 we have expanded a 2 hour interval between 10am and 12noon to demonstrate our key concepts.

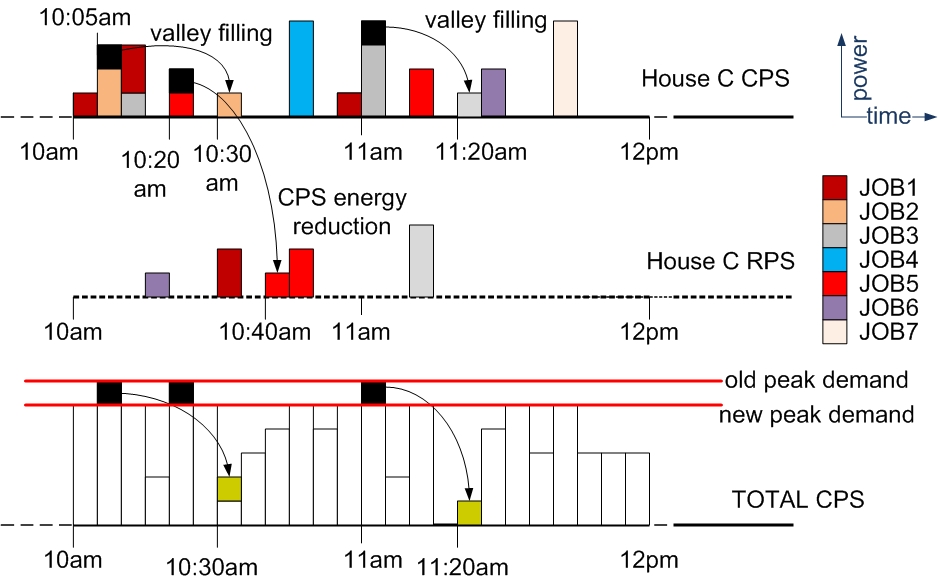
The horizontal arrows leading up to a job depicts the effect of scheduling that job from an OPT decision at an earlier time point. This decision might have been taken to reduce peak demand, or to take advantage of lower per unit cost of energy which was the pre-dispatch cost available to the OPT at an earlier point in time. Re-scheduling of a job from the current instance to a later one within the same hour can be done to reduce peak demand, and to a different hour to take advantage of a lower pre-dispatch price of power in a future hour compared to the real-time price of power in the current hour.

The power requirement of constant power (CP) loads are known a priori, while those of constant impedance (CI) and constant current (CC) loads depend on the voltage of the power supply at the point where these loads are connected. The CI and CC loads will be virtually treated as CP loads during the run of OPT.



**Figure 2: An example scheduling of loads in 3 houses on the same commercial feeder**

*Figure* 3 depicts an optimized schedule for house C (that can be achieved by its OPT). Here the CPS peak demand has been reduced by valley filling at two time instances, and CPS energy requirement has been reduced at one time instant when an appliance load could be shifted to the RPS. Since both the real-time and pre-dispatch price of power, along with the availability of RPS power change dynamically, a dynamic scheduling scheme is necessary. The simple scheduling optimization in *Figure* 3 reduces the peak CPS demand and achieves a net reduction in CPS energy demand. The cost of energy for RPS is $0. Slicing portions of vertical bar of a job and shifting them on the time scale, corresponds to rescheduling specific appliances mapped to that job at different time instances. Note that the entire power demand of a specific appliance has to be shifted in time. If a job has a single appliance mapped to it, the vertical bar (corresponding to its power) cannot be sliced, and will have to be shifted on the time scale as a unit.



**Figure 3: Load scheduling to reduce peak demand and energy from CPS**

OPT Algorithm

In this section we will describe the OPT algorithm which executes for every house in the community. The OPT falls in the class of constrained optimization problems which we will solve using a fast heuristic. Note that the OPT for every house will have to be solved and the house load scheduling result will have to be broadcasted (using some wireless network) to the utility server.

**Variables**

The assignment times of jobs onto different power sources, including the CPS and the set of renewable sources and storage units {RPSk}. In *Figure* 2 we have shown one such RPS for house C. Hence, there is a total of (k+1) power sources for a certain house.

**Constraints**

1. Each appliance has a total energy demand that must be satisfied over all its scheduled instances.
2. Each appliance has a time range of execution bounded by a start-time and a deadline. If the run time of the appliance and its time range of execution coincide, the appliance has no flexibility in scheduling. The final schedule achieved by OPT has to meet these constraints.

(2a) Some appliances {Jx} can be considered Soft Real-Time (SRT) constrained, and they do not have to meet strict deadlines. The tardiness of these appliances {D(Jx)}, which is measured by their completed delays beyond their deadlines, are part of the cost function which will be minimized.

1. Elements of {RPSk} have their own peak power generation and storage constraints.
2. The CPS can be peak power constrained as desired. In this work we are assuming infinite CPS capacity, but we are aiming to reduce the peak power demand, and therefore the cost of buying electricity for the consumers.

**Cost Function**

The total cost to be minimized is given by equation (1) below, where T$ is the cost of electricity from the CPS and the second term is the sum of tardiness of all SRT jobs {Jx}.

(1) (2)

Equation (2) explains the total cost of buying CPS electricity. KWh represents the total energy (in kilowatt-hour). U$energy is the unit price of energy; by shifting demand into the demand-free charge (valley), and thereby a lower rate, it is quite conceivable that U$energy is reduced.

**Simulated Annealing based Load Scheduling**

From *Figures* 2 and 3, it can be seen that our load scheduling problem has two dimensions:

1. Assignment of a job partition (corresponding to one or more appliances mapped to the job) to an instance (of duration 5 minutes) on the time scale, and
2. Assignment of such a job partition to one of the (k+1) power sources.

A fast and efficient placement algorithm that achieves good results for such 2-dimensional placement problems is Simulated annealing (SA), a Monte Carlo approach for minimizing multivariate functions.  The term SA derives from the roughly analogous physical process of heating and then slowly cooling a substance to obtain a strong crystalline structure. In simulation, a minima of the cost function corresponds to this ground state of the substance.  The SA process lowers the temperature by slow stages until the system “freezes” and no further changes occur.  At each temperature the simulation must proceed long enough for the system to reach steady state or thermal equilibrium.

The ***SA algorithm*** we use has four basic components:  
***Configuration***: A model of what a legal placement or schedule or configuration is (**Variables**).  This represents the possible problem solutions over which we will search for the optimal solution.   
***Perturbation***: A set of allowable moves that will permit us to reach all feasible configurations the algorithm proceeds. These perturbations have to be constrained by a set of **Constraints**. We consider the random re-placement of a randomly selected load anywhere on the 2-dimensional time-space continuum as described in (A) and (B) above, subject to its constraints.  
***Cost function***: To measure how *good* any given configuration is. We use equations (1) and (2) above to evaluate the cost function.  
***Cooling schedule***: To anneal the problem from a random solution to a good, frozen, configuration.  Specifically, we need a starting hot temperature (or a heuristic for determining a starting temperature for the current problem) and rules to determine when the current temperature should be lowered, by how much the temperature should be lowered, and when annealing should be terminated.

The self-explanatory pseudo-code for the SA algorithm is given below:

***Pseudo Code for SA***

*1. start with the system in an intitial know configuration & known energy E;*

*2. initial temperature T = Tinit; frozen = FALSE;*

*3. while (not (frozen)) {*

*4. repeat {*

*5. perturb system;*

*6. compute E = new value of cost function – old value cost function;*

*7. if (E < 0) {*

*8. Definitely accept this as the new system configuration;*

*9. }*

*10. else {*

*11. Accept this as the new system configuration with probability exp (E/T);*

*12. /\*Compute exp (E/T) and generate a random number r.  If r is less than exp (E/T), then keep it.  Otherwise, reject it.\*/*

*}*

*13. } until (system is in thermal equilibrium at T);*

*14. /\*for the last M attempts with constant T, there has been no reduction in cost function\*/*

*15. if (cost function has been decreasing over the last K temperatures) {*

*16. T = 0.9\*T; /\*cooling schedule\*/*

*17. }*

*18. else { frozen = TRUE; }*

*19. }*

*20. return the configuration with the lowest energy solution; /\*with best cost function value\*/*

### 1.3 Deliverables

The time varying day-ahead LMP (Locational Marginal Price) of electricity can be found at: <http://www.pjm.com/markets-and-operations/energy/day-ahead/lmpda.aspx>

Find the predicted price ($/MWhr) from the relevant csv file for a window of 24 hours, and use it for predictive scheduling.

Renewable power can be sold back to the utility during an hour’s window, at the same cost as the LPM for that hour. Any renewable power that is not stored or used by the user is automatically sold back to the utility. The utility credits the user’s monthly bill with the buy-back power cost.

A *COMMIT* is a user schedule change; such a scheduling change may be initiated by the user, or by the user’s acceptance of a dynamic rescheduling by the utility server.

The overall schedule at the utility is updated every time there is a *COMMIT* from any user. The updated power usage profile is then broadcasted to all users (while maintaining individual user privacies).

The scheduling optimization algorithm should be continuously running on the utility server, which has a view of ALL existing user schedules. Note that all user schedules are broadcasted to the utility if any change is made at all. The server may decide to notify certain users about the cost benefits of their schedule changes.

* If the user accepts all the changes, he makes a *COMMIT* to the utility server.
* If the user accepts parts of the changes, he can run a rescheduling algorithm on his home server/mobile device and notify the utility of the changes. The utility then replies to this user with cost implications. If the user accepts it, he makes a *COMMIT* to the utility server.

The user can decide to make changes to his schedule by

* altering schedule constraints on his appliances
* moving appliances over to his RPS
* moving appliances back to the CPS in case of RPS non-production or empty storage

The user can

* either submit a new set of constraints for his appliances to the utility and let the utility schedule for him, or
* run a rescheduling algorithm on his home server or mobile device, and regenerate a new schedule for himself (this has to be done if he is scheduling appliances onto his RPS)

The proposed changes are sent to the utility, which will run a scheduling and/or cost analysis of the changes and send the results back to the user. Based on this information, the user may or may not choose the changes he proposed. If he still goes ahead and chooses to make the changes, he sends a *COMMIT* to the utility, and the latter then updates the overall schedule accordingly.

All groups should research different ways of partitioning the work in the app (running on the mobile device), between a resource-constrained client (the mobile device) and a resource-rich home server, both of which are connected by a wireless network. The effectiveness of any job partitioning will be evident from measurements of key metrics such as response times, throughputs, energy usage, and performance bottlenecks, on both the mobile devices and on the servers. We use a profiler known as ***Trepn*** on the mobile devices, and Intel ***VTune*** or ***gprof*** on the servers running on Linux. Trepn is a profiling tool, developed by Qualcomm Inc, which profiles the performance, chip resource usage, and energy consumption of Android applications. The granularity of profiling by Trepn ranges from the whole system, to a specific app, to a specific function or thread in an app. Similar analysis will be done by VTune or gprof on the server side.

There are 4 codes to design: (a) the app on the mobile device, (b) the home server code, (c) the utility server code, and (d) the partitioner on the mobile app to distribute work between the mobile device and the home server.

The home server is assumed to be communicating with the appliances, and has the appliance models and schedules in it. This server is always directly accessible to the user’s mobile device. This home server should be implemented in a Virtual Box or Container.

The utility server should be implemented in a different Virtual Box or Container. Besides the user loads and schedules of the 3 group members and the 2 graders (all considered as different users), the utility server should have dynamically changing models of “other user loads” from a community of say 100-1000 other users. You have to generate these dynamic load profiles.

Below are some suggested starting point ideas for your mobile app – this is by no means complete. Design the rest of your app in consultation with your instructor. You will be graded based on your interactions with the instructor as a design and implementation team.

