CHALMERS | GÖTEBORG UNIVERSITY

Technical Report No. 2012-13

A greedy algorithm for the unforecasted energy dispatch problem with storage in Smart Grids

GIORGOS GEORGIADIS
MARINA PAPATRIANTAFILOU

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY/
GOTEBORG UNIVERSITY
Goteborg, Sweden, 2012

A greedy algorithm for the unforecasted energy dispatch problem with storage in Smart Grids

Giorgos Georgiadis, Marina Papatriantafilou
Department of Computer Science and Engineering, Chalmers University of Technology
S-412 96 Göteborg, Sweden
Email: {georgiog,ptrianta}@chalmers.se, Fax: +46-31-7723663

Abstract

Recent modernization efforts of the electrical grid led to the introduction of an additional communications and computations-based layer in the infrastructure, with the resulting new grid commonly referred to as *smart grid*. One of the aims of the smart grid is to facilitate the integration of renewable and distributed energy sources on a massive scale but these technologies bring benefits as well as challenges. Part of the challenge is due to intermittent energy sources, such as wind and solar, which generate energy at irregular intervals, leading to utilization problems for the grid. On the other hand, upcoming storage technologies, such as electrical cars, hold the potential to store and utilize this intermittent supply at a later time but bring challenges of their own, for example efficient storage utilization and intermittent energy demand.

In this paper we present an algorithm that enables distributed solutions by utilizing efficiently any storage capabilities in order to mitigate the effect of unreliable or non-existent demand forecasts. Drawing upon computer science methodology, we define and model the problem of unforecasted energy dispatch with storage as a scheduling problem of tasks on machines. We show that both storage and the time parameter inherent in the energy dispatch problem can be incorporated into a variant of the scheduling problem, leading to a novel modeling that can be used to study further the energy dispatch problem. In addition to presenting a simple but effective algorithm that solves the aforementioned problem, we also prove analytically that this is done in a near optimal way. Finally, we provide an extensive simulation study for a variety of scenarios, showing that the presented algorithm is highly competitive to methods that use forecasts and assume total knowledge about the demand requests.

1 Introduction

In recent years there has been an organized effort on an international level to modernize the power grid by adding resilience properties, precise accounting and new services though the use of information technologies, collectively leading to a new type of grid commonly called *smart grid*. On the one hand these changes address a much needed modernization of the aging grid, for example energy loses due to ineffective infrastructure and energy theft. On the other hand, it is expected that these changes will enable the incorporation of renewable (e.g. photovoltaic arrays and wind generator farms) and distributed (e.g. electric car fleets) energy sources on a large scale. Nevertheless, these new generation technologies bring benefits as well as challenges.

Currently there are established models and methods (cf [9] and bibliography therein) that can *forecast* quite accurately the expected consumption on the power grid over the course

of 24 hours, especially since consumers follow distinct diurnal patterns, and meet it with adequate supply. However, the intermittent nature of renewable energy sources such as wind generator farms challenges the way we utilize energy when it is available, compensate for when it is not and rely on weather forecasts for the grids' daily operation. On the other hand, distributed energy sources such as electric car fleets can act as storage options and balance the demand and supply of electrical energy but bring challenges of their own, for example efficient storage utilization and intermittent energy demand. Since options for adapting or forecasting the generation capabilities of these energy sources are currently limited, we focus on solutions that adapt the demand and use storage capabilities to mitigate their intermittent effects.

In the present work we model the problem of energy dispatch from producers to consumers in the distribution level of the grid, using an extended *online load balancing problem*. By drawing from the computer science methodology on scheduling tasks to machines, we present a novel scheduling and resource dispatching algorithm that is able to cope with the inherent unpredictability of renewable energy sources without the use of forecasts, as well as take advantage of available storage options in the grid. Note that the aim is not to ignore the possible benefits of forecasts but to show the benefits of alternative methods when coping with uncertainty, e.g. when demand requests arrive in an arbitrary way (commonly referred to as *online*) or when contingency plans must be drawn for the grid's correct operation . At the core of our modeling lies the transformation of the flexible load demand requests' time parameter (i.e. at which time does the requested energy need to be delivered) into a restricted set of allowed machines in the task scheduling domain, as well as the incorporation of storage in the optimization criteria. We also show analytically that the presented algorithm can make efficient use of storage (if present) and is near optimal for the specific problem. Finally we conduct an extensive simulation study for a variety of scenarios and show that the presented algorithm is highly competitive to methods that use forecasts and assume total knowledge about the incoming load demand requests for the same problem.

2 Problem background and definitions

System and problem definition We focus on the distribution system of the grid as a high level abstraction. This includes a number of *nodes* being connected through a simple topology, for example low voltage power lines in a radial feeder configuration [13] that connects all nodes to the transmission system (see figure 1 for an example). These nodes issue *load demand requests* on the feeder at irregular intervals, independently from each other, and energy is being dispatched from generation sites to satisfy the demand. Note here that energy dispatch can be coordinated by a central authority or be issued locally by a power aggregator on a local distribution grid level. Since the algorithm works independent from this parameter, as long as the interface is the same, this distinction is going to be disregarded in the rest of the paper.

The load demand requests are commonly simple loads that consume energy at the moment they are requested, for example turning on the lights in a room. However, some of them may be able to delay the energy consumption requested for a period of time, for example laundry machines that can be programmed to start at a later time. Other requests may be able to store energy in the system in some form, for example refrigerators that run part of their cooling cycle at an earlier time (precooling). These two request characteristics are com-

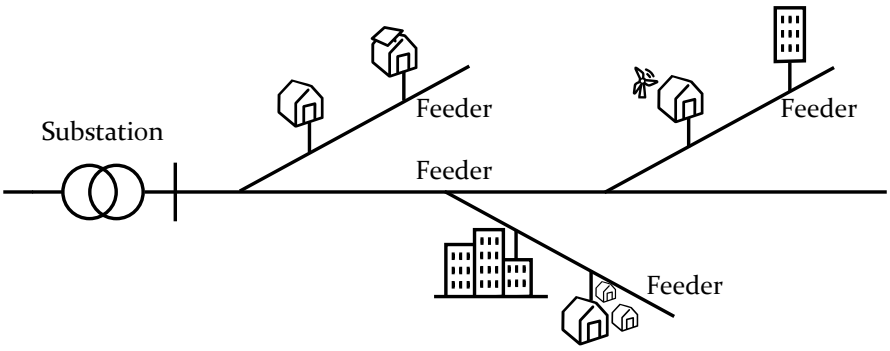

Figure 1: High level schematic representation of an example distribution level subgrid

monly called *slide* and *slack* respectively. Finally, some requests may be simply issued for energy that will be stored in order to be used at a later time, for example charging electrical vehicles' batteries. We call these *storage* requests. Note here that, regardless of its specific properties, every load demand request must be scheduled at the time of its issuing although it can be served at the same or later time (depending on its type). Based on whether the stored energy can be accessed only locally (on the node it was stored, i.e. a households' heat pump) or globally (by any node in a distribution system subgrid), it can be characterized as *local* and *global* storage respectively. In the continuation we will refer to slack as storage requests, even though they are based on a different mechanism, since they are similar in their effects.

Adding to the complexity of energy dispatch, combined heat and power (CHP) plants offer *thermal energy dispatch* along with electrical energy as part of their services. The thermal energy these power plants produce can be used for a number of devices that would normally require electrical energy (for example heating/cooling services) and can even be stored in order to be used at a later time, for example in heat pumps. In the following sections we are considering both electrical and thermal services offered in two separate feeder lines on all nodes, an electrical and a thermal one respectively.

In the rest of the paper we focus on the *unforecasted energy dispatch problem with storage*, defined as follows:

Definition 1 (Unforecasted energy dispatch problem with storage) *Given a distribution system subgrid, we call* unforecasted energy dispatch problem with storage *the problem of dispatching generated electrical and thermal energy to end consumers without using forecasts and by taking into account any storage capabilities present, while trying to minimize peak consumption.*

Computing problem A similar problem to the energy dispatch problem mentioned above, originates in the parallel processing community of computer engineering and is framed in terms of machines (i.e. processors, computers) and tasks (i.e. threads, computer programs). Consider a number of machines and tasks that we want to schedule to run on these machines. The tasks are incurring a permanent load on the machines that are running and we have no information on the total number of tasks, the tasks' load or their time of arrival, i.e. the tasks are arriving at arbitrary intervals (*online*). The problem of scheduling such tasks to machines in a way that minimizes the maximum load on any machine is called the *online*

load balancing problem. Depending on whether all tasks can run equally well on all machines, with performance differences or only on specific machines, we can further characterize the problem as having *identical*, *related/unrelated* or *restricted* machines respectively. Of particular relevance here is the third category, since the additional constraint to restrict tasks into running only on specific machines fits our energy dispatch problem quite well, as we are going to show below.

3 Modeling and transformation

Modeling details Following the description of the problem presented in the previous section, we identify three orthogonal axis to characterize load demand requests:

> **elastic/inelastic**, according to the ability to shift the demanded load over time or not, respectively. Elastic loads can be scheduled to be serviced within a set of timeslots, while inelastic loads must be serviced necessarily in a specific time slot.
>
> **electrical/thermal**, depending on whether the demanded load can be serviced using only electrical energy or both electrical and thermal, respectively.
>
> **storage/simple**, according to the ability to store the demanded load for future use or not, respectively[1].

We are going to disregard the distinction between local and global storage since our theoretical results apply equally to both types. For the same reason and for simplicity in the experimental section we treat all storage as global. The latter choice is further motivated by practical applications, since local storage can be made available on a global level as part of an additional service layer offered by the utility or aggregator (for example batteries of electric vehicles acting as distributed energy sources and providing energy for their immediate neighborhood). For reasons of convention, throughout the rest of the paper we are assuming that requests are scheduled in hourly timeslots and are coming in a diurnal pattern, i.e. each request must define its allowed set (possibly a singleton) of hourly timeslots to be scheduled in and the timeslots are labeled from 00:00 to 23:00, corresponding to the starting point of the hour that they refer to.

Problem transformation We propose the transformation of the unforecasted energy dispatch problem with storage defined above into an extended online load balancing problem by expressing load demand requests as an input sequence of tasks $t_0, \ldots, t_j, \ldots$ to be run on machines and the electrical and thermal feeder lines as the machines themselves $M_i, i = 0 \ldots n-1$ (figure 2). The additional restrictions about the usage of forecasts and the existence of storage capabilities are accommodated as follows.

> **No forecasts** The fact that no forecasts are being used about the incoming requests corresponds to the online property of the load balancing problem: the tasks are arriving at arbitrary intervals and no information exists about them prior to their arrival. The dimension of time associated with requests (i.e. an elastic request can be scheduled in any of a set of timeslots) is expressed by creating a copy of each electrical and thermal

[1]Note that in the continuation we will omit the "simple" characterization, implying that all loads will be simple unless otherwise specified.

feeder line for every timeslot. By adopting a diurnal and hourly scheduling period we reach a total of $n = 48$ machines (24 electrical and 24 thermal, for convenience numbered $M_0, \ldots, M_{\frac{n}{2}-1}$ and $M_{\frac{n}{2}}, \ldots, M_{n-1}$ respectively), upon which the tasks can be run.

Storage When a storage task runs on a machine it generates a load on the specific machine but also a *load credit* that can be used in subsequent task runs on the same or "future" machines of the same kind. This is the case since unused energy at a specific timeslot is available in later timeslots, although it can be available in diminished amounts due to dissipation or low energy conversion efficiency. Note that in our experiments we consider thermal energy to dissipate exponentially (meaning that less energy is available in subsequent timeslots) while electrical energy does not dissipate. This is done for convenience reasons and is not a limitation of the algorithm, since our theoretical analysis applies the same when the algorithm operates with or without dissipation or any kind of losses for any energy carrier. For the same reason we consider storage tasks to be elastic.

We call the resulting problem from the above transformation an *online load demand balancing problem with storage*, which is defined as follows:

Definition 2 (Online load demand balancing problem with storage)
Let $M_i, i = 0 \ldots n-1$ be a set of machines where variable load credit (i.e. storage) can accumulate and $t_0, \ldots, t_j, \ldots$ be an input task sequence of two task types, simple and storage, with the following properties: each task t_j of both types has load w_j and restrictions on the allowable machines it can run on, while storage tasks additionally create on all machines load credit equal or less to their load (with the possibility of 0 on some but not all machines[2]). We define the online load demand balancing problem with storage *as the problem of assigning the tasks to the machines while minimizing the maximum load on the machines.*

An example instance of the transformed problem can be found in schematic form in figure 2.

4 A Greedy Algorithm

Our starting point is the known, greedy algorithm (which we will refer to simply as GREEDY) from online load balancing for machines and tasks:

Definition 3 *[4] The* GREEDY *algorithm assigns each incoming task t_j to the machine that, after being assigned t_j will have the least load (breaking ties arbitrarily).*

Based on the modeling that we propose in Section 3, we adapt the GREEDY algorithm and we augment it to support storage, getting the STORAGEGREEDY algorithm presented here:

Definition 4 *The* STORAGEGREEDY *algorithm assigns each incoming task to the allowed machine having the minimum load-storage difference (ties are broken arbitrarily).*

In practice, the above definition can be expressed as follows: on a system level and for a load demand request with specific restrictions, schedule the request to be executed on the

[2]The trivial case of load credit equal to 0 on all machines corresponds to simple tasks.

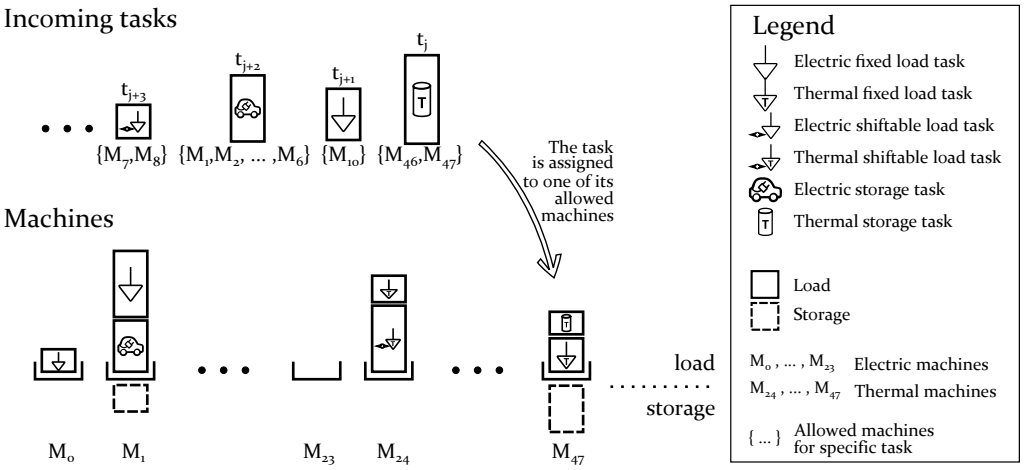

Figure 2: An example of the online load balancing problem for electrical and thermal energy requests with storage capabilities

allowed timeslot with the minimum load-storage difference. Note here that the scheduling is taking place on the node that issued the request, where a copy of the algorithm is running. These algorithmic copies run autonomously, with sole knowledge of the publicly available scheduled totals of load and storage for different timeslots and for the different energy carriers (electrical/thermal). We assume that loads are being registered to a timeslot as soon as they are assigned to it and that the scheduled totals of load and storage are maintained in a way that keeps them consistent with simultaneous reads/writes by the nodes (i.e. the load/storage values are always correct). Figure 3 below shows an example execution of the algorithm.

The pseudocode of the algorithm running at each node can be found in algorithm 1 below, along with a brief description of the variables and functions used. Note that all variables are local except `loadMachines` and `storageMachines` which are common for all nodes and contain respectively the load and storage of all machines.

Variables:

`loadMachines, storageMachines` Common variables (arrays) of all machines for load and storage respectively. As mentioned above, we assume that all nodes have access to the same information regarding load and storage content of the machines and a mechanism exists that keeps their values consistent during simultaneous reads and writes by the nodes.

`taskId` The unique identifier of the task to be scheduled by the node.

`machineId` The unique identifier of the machine with minimum load-storage difference, as returned by the function `minMachine()`.

`storTaskId, storageValue` The unique identifier of the task that stored the energy to be used and the actual value of the stored energy respectively, as returned by the function `findStorageTask()`.

`remainingLoad, remainingStorage` Intermediate variables containing the remainder of the task's load and machine's storage respectively, while the task is being assigned

to the machine.

Functions:

`minMachine()` Returns the identifier of the machine with minimum load-storage difference.

`findStorageTask()` Returns the identifier of a task that stored energy on a specific machine. The selection process currently uses the heuristic rule of selecting the last task that stored energy on the machine, although any number of rules can be implemented.

`consumeStoredEnergy()` Consumes the energy originally stored by task `storTaskId`.

`produceStoredEnergy()` Runs if `taskId` is a storage task and stores the appropriate amount of energy, starting on machine `machineId` and continuing on subsequent machines.

`getAllowedMachines()` Helper function that returns the set of allowed machines for a given task.

`getTaskLoad()` Helper function that returns the load of a task.

As mentioned in the previous section, our algorithm can operate both with dissipating and non-dissipating energy storage. In the case of non-dissipating storage, a storage task that is run on a machine generates the same amount of storage on the same and subsequent machines of the same type. In the case of dissipating storage, the amount of energy stored in subsequent machines is less according to the appropriate dissipation rate (the further a machine from the one the task run on, the less stored energy is available). When a task uses energy stored on a machine, the appropriate amount is being subtracted from all machines of the same type (either the same or diminished according to the appropriate dissipation rate, in the case of non-dissipating and dissipating storage respectively), to reflect the fact that this stored energy is no longer available for subsequent tasks.

Algorithm 1 Assign an incoming task *taskId* to the machine with minimum load-storage difference

Input: $loadMachines, storageMachines, taskId$

Output: $loadMachines, storageMachines$

```
machineId ← minMachine(loadMachines,storageMachines,getAllowedMachines(taskId))
remainingLoad ← getTaskLoad(taskId)
remainingStorage ← storageMachines[machineId]
while (remainingLoad > 0) ∧ (remainingStorage > 0) do
    storageTaskId, storageValue ← findStorageTask(machineId)
    consumeStoredEnergy(remainingLoad,storageTaskId)
    remainingLoad ← remainingLoad − min(remainingLoad,storageValue)
    remainingStorage ← remainingStorage − storageValue
loadMachines[machineId] ← loadMachines[machineId] + remainingLoad
if loadMachines[machineId] is type(storage) then
    produceStoredEnergy(machineId,taskId)
```

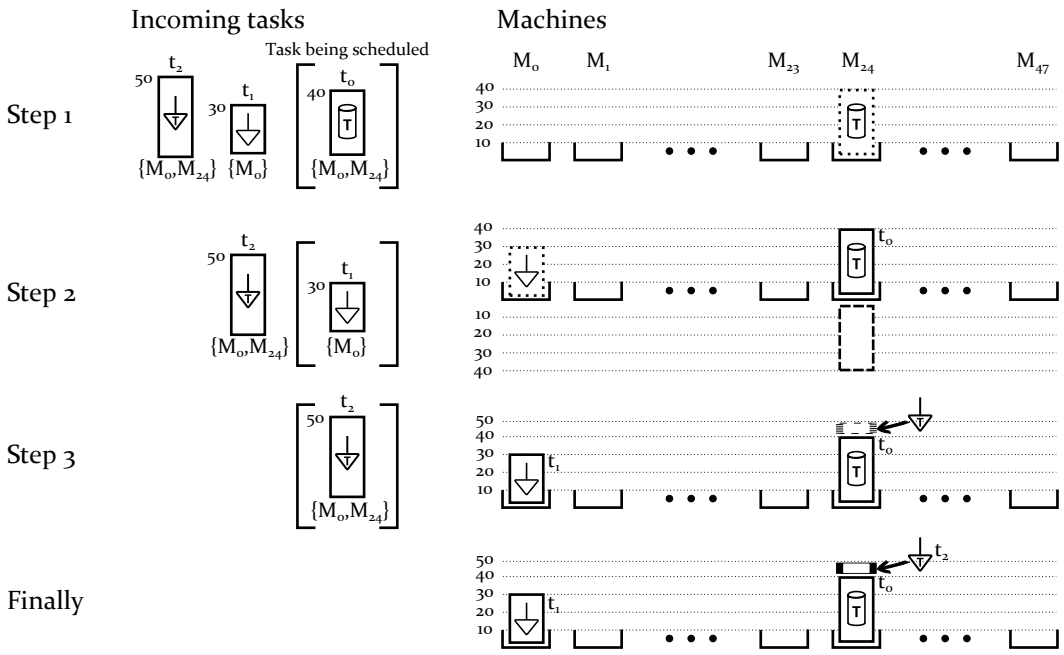

Figure 3: An example run of the StorageGreedy algorithm (task loads shown at the left side of each task). Step 1: the task scheduler places the first task to machine M_{24}. Step 2: the second task is placed to the only allowed machine M_0. Step 3: between the two allowed machines M_0 and M_{24}, the one having minimum load-storage difference is M_{24}. The scheduler places the task there, uses up all available storage and adds the remaining load to the machine.

5 Theoretical analysis

In this section we follow a methodology similar to [4] and show that the performance of the StorageGreedy algorithm is near optimal when compared with the optimal algorithm OPT for the load demand balancing problem with storage. The appropriate metric for this comparison is the *competitive ratio*, which is defined in the literature as follows.

Definition 5 *[4] An online algorithm* ALG *has a* competitive ratio c*, compared to the optimal algorithm* OPT *for the same optimization problem, if there is a constant* α *such that for all finite input sequences* I *the following holds true for the solution costs* $ALG(I)$ *and* $OPT(I)$*:*

$$ALG(I) \leq c \cdot OPT(I) + \alpha$$

The following theorem shows that the solution cost of our algorithm lies within a factor of $\lceil \log n \rceil + 1$ when compared to the cost of the optimal algorithm, where n is the number of machines (according to our modeling, $n = 48$). Note that no assumptions are made regarding the dissipation of the stored energy and as a result the theorem applies both for dissipating and non-dissipating storage.

Theorem 1 *Algorithm* StorageGreedy *achieves a competitive ratio of* $\lceil \log n \rceil + 1$ *for the online load demand balancing problem with storage, where* n *is the number of machines.*

Proof: Let $\sigma = u_1, u_2, \ldots, u_m$ be a sequence of tasks u_k, each having load w_k, with total load $W = \sum_{k=1}^{m} w_k$. We define $l = OPT(\sigma)$ be the maximum load on the machines achieved by the optimal algorithm OPT for that sequence. Clearly,

$$l \geq \frac{\sum_{k=1}^{m} w_k}{n} = \frac{W}{n}$$

Also let M be the assignment of the aforementioned task sequence to machines, as generated by the StorageGreedy algorithm. To facilitate the proof we will partition M into layers of size exactly l (see figure 4), by observing that each task can cross over layers at most once. After using up the total stored energy that was available (including the possibility of zero storage), each machine either has a workload of exactly l in a layer, implying that it continues to run at the next layer, or less than l, in which case we conclude it finished running tasks and will never resume. We will prove that there can be at most $\lceil \log n \rceil + 1$ such layers needed by the StorageGreedy algorithm, which will give the desired competitive ratio.

Let W_i be the sum of the weights of the tasks in layer i as assigned by the StorageGreedy algorithm and let R_i be the total load not yet assigned by StorageGreedy up to and including layer i. It is easy to see that $W_i = R_{i-1} - R_i$. To prove the above mentioned competitive ratio it is enough to show that $W_i \geq R_i$ for every layer i, since it then follows that

$$R_i \leq R_{i-1} - R_i \Rightarrow R_i \leq \frac{R_{i-1}}{2}$$

and by solving the recursion we get

$$R_{\lceil \log n \rceil} \leq \frac{R_0}{n} = \frac{W}{n} \leq OPT\,(\sigma)\,.$$

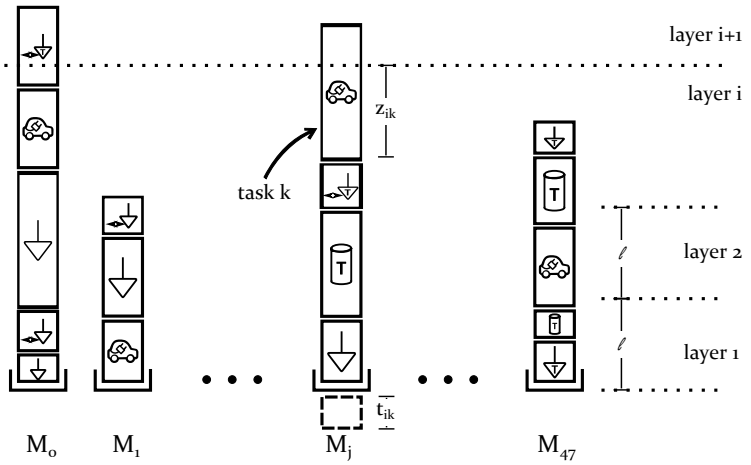

Figure 4: An example partitioning of assignment M into layers. By t_{ik} we denote the available storage at machine M_j at the beginning of task's k assignment and by z_{ik} the assigned load of task k after the consumption of storage t_{ik} and before layer i was completed.

Any load remaining after layer $logn$ can be assigned at the next layer $logn + 1$, which gives us the desired competitive ratio. Next we provide the notation necessary for the remainder of the proof and we prove the inequality $W_i \geq R_i$ for every layer i.

Notation and definitions:

- W_{ij} is the load of machine j at layer i. Note that $W_i = \sum_{j=1}^{n} W_{ij}$.
- z_{ik} is the (partial) weight of task k assigned at layer i and contributes in the load of the layer. Note that for every task k it can be $z_{ik} > 0$ in at most two successive layers.
- t_{ik} is the (partial) weight of task k assigned at layer i that runs using storage and therefore does not contribute in the load of layer i.
- O_j is the set of tasks that the optimal algorithm OPT assigns to machine j.
- O_{ij} is the subset of O_j that the StorageGreedy algorithm has not finished assigning or running at the end of layer i.
- R_{ij} is the total remaining weight of the tasks in O_{ij} (excluding any partial weights z_{ik} that might have been assigned in layer i). Note that $R_i = \sum_{j=1}^{n} R_{ij}$. Since R_{ij} is the total weight of a subset of O_j it also follows that $R_{ij} \leq l$.

The desired inequality holds when $O_{ij} = \emptyset$ or $W_{ij} = l$ since then we have $R_{ij} = 0 \leq W_{ij}$ and $R_{ij} \leq l = W_{ij}$ respectively. Below we examine the case $O_{ij} \neq \emptyset \wedge W_{ij} < l$.

Since $W_{ij} < l$ we conclude that machine j has finished running tasks and will not run in subsequent layers. However, the fact that $O_{ij} \neq \emptyset$ implies that there are tasks remaining which the OPT algorithm would assign to machine j but now need to run on other machines. Let k be such a task that belongs to O_{ij} but runs on machine r instead of machine j, and let S_r and S_j be their storage capabilities respectively at the time of the assignment. Note here that task k must increase the load of machine j up to l for layer i and continue its execution

past layer i since $k \in O_{ij}$. This leads us to conclude that $z_{ik} > 0$ and $t_{ik} = S_r$, where z_{ik} and t_{ik} are the partial weights of task k that contribute to the load of layer i or not respectively.

In order for task k to be assigned by the StorageGreedy algorithm to machine r (instead of machine j), its load-storage difference at the time of assignment must be less than or equal to the corresponding load-storage difference at machine j. Note here that both machines worked fully in the previous $i-1$ layers, otherwise they would have stopped working already by the time they reached layer i. Therefore, the criterion that must be satisfied is the following:

$$\begin{aligned} (i-1)\,l + W_{ij} - S_j &\geq (i-1)\,l + (l - z_{ik} - S_r) \\ W_{ij} &\geq l - z_{ik} - t_{ik} + S_j. \end{aligned} \tag{1}$$

On the other hand, by the definition of R_{ij} we have

$$\begin{aligned} R_{ij} = \sum_{k \in O_{ij}} w_k - z_{ik} - t_{ik} &= \sum_{k \in O_{ij}} w_k - \sum_{k \in O_{ij}} z_{ik} - \sum_{k \in O_{ij}} t_{ik} \\ R_{ij} &\leq l - \sum_{k \in O_{ij}} z_{ik} - \sum_{k \in O_{ij}} t_{ik}. \end{aligned} \tag{2}$$

Since $S_j \geq 0$, from eq. 1 and 2 we get $W_{ij} \geq R_{ij}$ and by summing for all machines we finally get $W_i \geq R_i$, which proves the desired competitive ratio. □

6 Experimental evaluation

The experimental evaluation study was conducted using the Swedish load demand mix for households [14], specifically the weekly mix for a four person family living in an apartment, that defined the different types of loads for a typical household along with their partial contribution in the daily total consumption. The data were elaborated further through the use of four different energy consumption profiles corresponding to 24 hour periods, selected from [9] as representative of the typical consumption patterns in distribution networks (figure 5). This allowed us to model different households (customer profiles) and different types of loads (i.e. elastic/inelastic, storage etc), as well as extrapolate synthetic data that were used as follows.

First, a number of scenarios were created along two axis: single or many households and flexibility of load with/without storage capabilities. The first axis is of particular interest since the algorithm can operate either on a household level (balancing the demand generated within) or on a local distribution network level (balancing the demand of many households). Subsequent scenarios were created with either a single household or four households using a mix of the four consumption profiles. The second axis is important in practice since the number of flexible loads and storage capabilities is currently limited but is expected to grow in the near future. Since our algorithm can operate with any number of these elements, we created three scenarios: a business-as-usual scenario, a moderate growth scenario and a full smart household/neighborhood scenario (see figure 6). In the business-as-usual scenario we assume all loads are simple electrical or thermal loads, with the exception of a small percentage of shiftable electrical loads (e.g. laundry machines). In the moderate growth scenario we included more shiftable electrical loads (e.g. refrigerators) and some thermal storage (e.g. heat pumps). Finally, in the full smart household/neighborhood scenario we

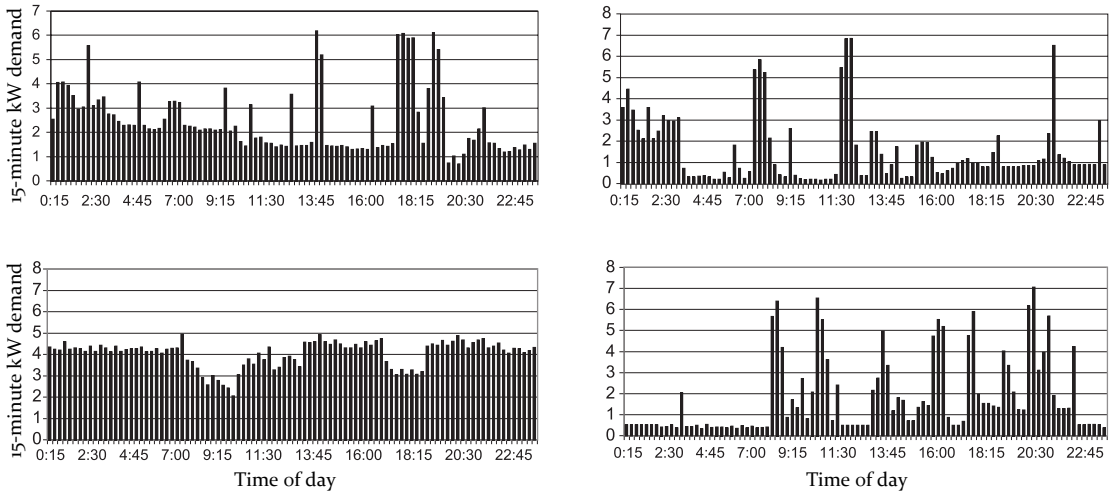

Figure 5: Different customer profiles used in creating synthetic data, as expressed by their diurnal consumption patterns [9]

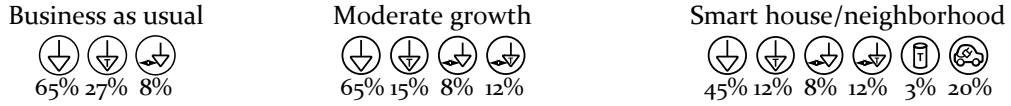

Figure 6: Types of loads considered for the experimentation scenarios, as percentages on the daily total

have included a fair amount of electrical storage (e.g. electrical cars) in addition to the moderate growth modifications.

During our experiments we aim at minimizing the peak consumption and focus on the comparison between our StorageGreedy algorithm and a simple, yet powerful algorithm for the same problem when forecasts are used, called the LPT algorithm:

Definition 6 *[6] For any finite input sequence of tasks, the* Longest Processing Time *(*LPT*) algorithm sorts them by decreasing processing time and then assigns each task to the machine that has the least load (breaking ties arbitrarily).*

Note that the theoretical analysis of Section 5 compares the StorageGreedy algorithm to the optimal algorithm OPT. However, since the OPT algorithm can be inappropriate due to prohibitively computational overhead [4], we conduct the experimental study presented here by comparing to the LPT algorithm, which is simple to implement and, for our problem, produces solutions of comparable quality with complex, state-of-the-art algorithms [7].

The full schedules from the two algorithms for all scenarios can be found in figures 7 and 8 below. For reasons of brevity we present here the multiple household case only for the smart neighborhood scenario (figure 8) since the results were very similar with the single household case for the other scenarios (due to the lack of storage capabilities).

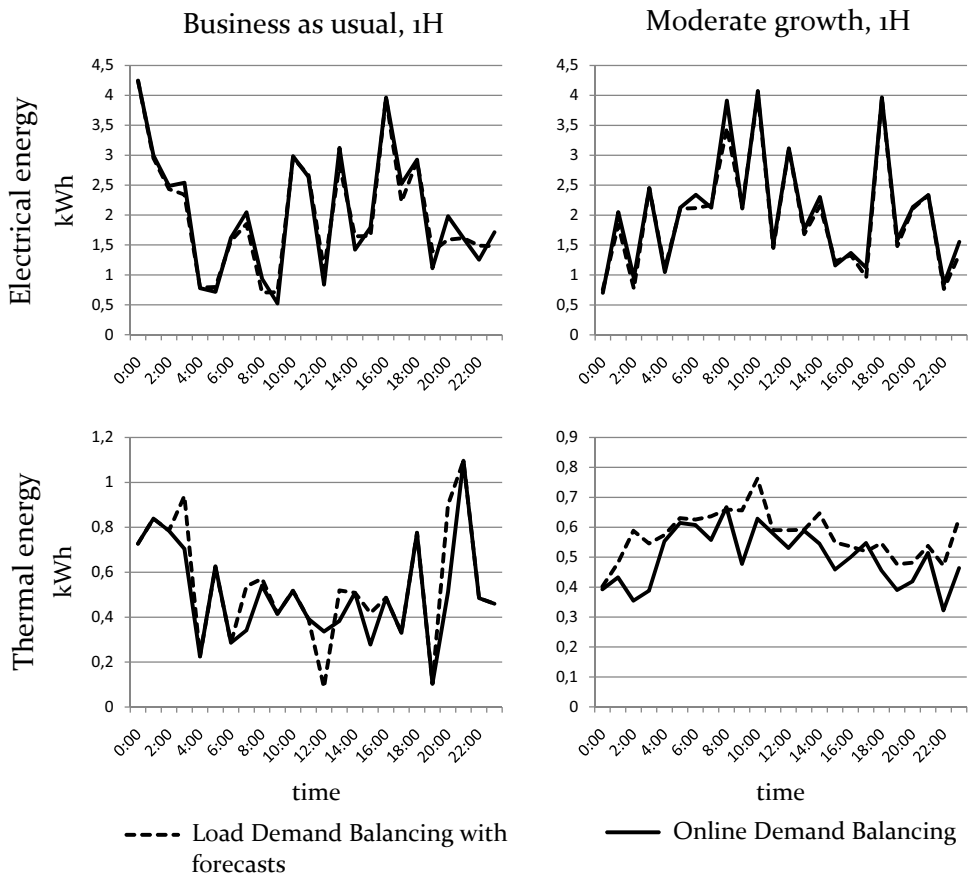

Figure 7: Comparison of load demand curves for the *Business as usual* and *Moderate growth* scenarios for one household

It is easy to see that for the business as usual and moderate growth scenarios there are very small differences between the two algorithms. This is the case even though the StorageGreedy algorithm has no access to forecasts and had to schedule incoming tasks immediately. Furthermore, it shows to be competitive with algorithms that use forecasts despite the lack of storage capabilities. Note here that the thermal energy consumption is a lot less than the electrical energy consumption on all scenarios and any differences there do not affect the end result.

In the smart household/neighborhood scenario the two algorithms have the opportunity to make quite different choices due to the presence of storage. Indeed, the resulting schedules differ to a greater extent but the StorageGreedy algorithm succeeds in lowering peak consumption by taking advantage of storage. As a result, its overall schedule is smoother in both the case of a single household and in the case of a four household neighborhood, even though the latter is smoother due to the different consumption profiles used there balancing out. Overall, during our experiments the StorageGreedy algorithm showed to be efficient without using forecasts and being able to take advantage of storage capabilities, if present, in order to further minimize peak consumption.

Figure 8: Comparison of load demand curves for the *Smart house/neighborhood* scenario for one and four household(s)

7 Discussion and related work

A number of generic computing methods have been proposed in the literature to address the energy dispatch problem but they are not directly comparable with the one presented here since they differ significantly in their models and focus areas. For example, Gaing in [5] use a particle swarm optimization heuristic method to solve the energy dispatch problem but choose focus on its economic aspects. Similarly, Koutsopoulos and Tassiulas in [10] address the same problem both in the offline and the online variations, but for the latter consider a stochastic dynamic model for the incoming demand requests in order to study timing related issues. Such a model differs significantly from ours, since for example a controller may choose to defer the dispatch to a later time, and it is not directly comparable.

In the computer science literature, the online load balancing problem and its generalization, the online task scheduling problem, are well studied problems and many algorithms exist for them (cf [4] and bibliography therein). Such algorithms include, for example, the ones suggested by Shmoys et al [12] and Albers [1], as well as the one by Graham [6] used here (cf also recent analysis and improvements by Johannes [7]). To the best of our knowledge, no modeling of the energy dispatch problem exists, that is based on the online load balancing problem. There are however recent approaches that address the online property of

the energy dispatch problem, such as the ones by Johnson et al [8], Narayanaswamy et al [11] and Barker et al [3]. The first two approaches make similar assumptions regarding the energy dispatch problem with the ones considered here and they both study the offline as well as the online variation of the problem. However, they both consider only one energy carrier (i.e. electricity), the presented solutions are not distributed and they do not model the problem in a generic setting as presented in Section 3. Furthermore, only Johnson et al in [8] include storage in their solutions but they consider only constant losses in addition to the lossless case, in contrast with the multiple types of losses considered here. In particular, Johnson et al provide analytical results that show their online algorithms to be H_n-competitive under certain conditions. In fact, the present paper addresses one of the open problems suggested by them, namely the analytical proof of competitiveness for online algorithms, albeit for a more generic setting than the one considered there. On the other hand, Narayanaswamy et al [11] focus on cost-based optimization using convex programming techniques and show that online algorithms for both offline and online variations perform better than certain "natural" algorithms found in literature. Finally, Barker et al [3] focus on a practical study that examines the effects of a Least Slack First online algorithm, which schedules first the elastic loads having the smallest allowed machines set. While not providing analytical results for their algorithm, they show that in a smart house with minimal amount of elastic loads it is possible to reduce demand peaks by over 20% in some cases.

The algorithm proposed here has a number of properties that render it particularly suitable for use in a smart grid context:

Enables autonomy, collaboration and increases the grids' responsiveness Each node can make its own decisions using locally available information, increasing in this way the overall responsiveness of the grid to unexpected changes such as a possibly catastrophic failure. However, two or more nodes can communicate, e.g. via Zigbee [2] or other local network smart grid equipment, in order to schedule their requests in a more efficient way. For example, when both have the option to schedule their requests at different timeslots or the same timeslot, they could coordinate to schedule in different timeslots in order to avoid a consumption peak.

Offers a demand-side management solution and increases the grids' adaptability The main function of the algorithm is to make efficient use of stored energy, as well as mitigate the effects of unpredictability in energy generation. This allows intermittent energy sources such as renewable or distributed energy sources to be fully utilized, leading to an increasingly adaptable grid where excess energy supply is utilized and possibly stored and where excess energy demand is served efficiently in an optimal way. Furthermore, the efficiency of the method leads to reduced consumption peaks which in turn lessen the need for overprovisioning and installation of expensive backup energy supply.

Parallel usage of both existent and future smart grid elements Since nodes operate in an autonomous way, they can employ smart grid technologies supported by the presented algorithm, such as distributed energy sources based on energy storage, independently of other nodes and without affecting or being affected by them. This means that both existent and smart grid technologies can be operational at the same time, making the transition to a fully smart grid as smooth and natural as possible.

8 Conclusions

In this paper we presented a novel modeling and an efficient algorithm for the unforecasted energy dispatch with storage problem in smart grids. We showed that both storage and the time parameter inherent in the energy dispatch problem can be incorporated into a variant of the scheduling problem of tasks on machines, leading to a novel modeling that can be used to study further the energy dispatch problem. We also presented a simple but effective algorithm that can utilize efficiently any storage capabilities in order to mitigate the effect of unreliable or non-existent demand forecasts, as well as proven analytically that this is done in a near optimal way.

We complement our modeling and algorithm with an extensive simulation study for a variety of scenarios, differing on two main axis: the consideration of one or more households and the amount of flexible loads as well as their type. We found that the new algorithm is highly competitive to methods that use forecasts and assume total knowledge about the demand requests, giving solutions of similar or greater quality in all cases. In particular, the new algorithm preformed better when the amount of flexible loads and storage options was increased, which was expected since it is designed to take advantage of these options if present. The performance of the new algorithm in the case of multiple households was similar with the single household case except for a smoother load demand curve, since the different customer profiles used for the creation of the households balance each other out, up to a degree.

The modeling presented here could be extended and applied to a wide range of energy dispatch problems, which we intend to address in future work. In this work we assumed implicitly an infinite amount of supply (all demands are satisfied) but the same modeling could be applied in cases where supply can be forecasted up to a degree and the unforecasted demand must adapt to it by necessity (e.g. according to received price signals). Moving in the same direction, it is possible to extend the present modeling in order to take into account the case of online supply in addition to online demand. In this case, supply and demand not only must be matched to each other as before but also in real time, without any forecasts. An algorithm that addresses this problem can be the core of an advanced demand shaping service, able to connect microproducers and microconsumers of electrical energy on a one-to-one basis and in real time, leading to minimization of the storage facilities needed in the system and to maximum utilization of renewable and distributed energy sources.

References

[1] S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, January 1999.

[2] ZigBee Alliance. ZigBee specification. 2007.

[3] S. Barker, A. Mishra, D. Irwin, P. Shenoy, and J. Albrecht. SmartCap: flattening peak electricity demand in smart homes. In *2012 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 67 –75, March 2012.

[4] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, April 1998.

[5] Z.-L. Gaing. Particle swarm optimization to solving the economic dispatch considering the generator constraints. *IEEE Transactions on Power Systems*, 18(3):1187 – 1195, August 2003.

[6] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, March 1969.

[7] B. Johannes. Scheduling parallel jobs to minimize the makespan. *Journal of Scheduling*, 9(5):433–452, 2006.

[8] M. P. Johnson, A. Bar-Noy, O. Liu, and Y. Feng. Energy peak shaving with local storage. *Sustainable Computing: Informatics and Systems*, 1(3):177–188, September 2011.

[9] W. H. Kersting. *Distribution System Modeling and Analysis*. CRC Press, 1 edition, August 2001.

[10] I. Koutsopoulos and L. Tassiulas. Control and optimization meet the smart power grid: scheduling of power demands for optimal energy management. In *Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking*, e-Energy '11, page 41–50, New York, NY, USA, 2011. ACM.

[11] B. Narayanaswamy, V. K. Garg, and T. S. Jayram. Online optimization for the smart (micro) grid. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, e-Energy '12, page 19:1–19:10, New York, NY, USA, 2012. ACM.

[12] D.B. Shmoys, J. Wein, and D.P. Williamson. Scheduling parallel machines on-line. In *, 32nd Annual Symposium on Foundations of Computer Science, 1991. Proceedings*, pages 131 –140, October 1991.

[13] A. von Meier. *Electric Power Systems: A Conceptual Introduction*. Wiley-IEEE Press, 1 edition, July 2006.

[14] J. P. Zimmermann. End-use metering campaign in 400 households in sweden assessment of the potential electricity savings. *Energimyndigheten, Sweden*, 2009.