Dynamic Scheduling for Energy Minimization in Delay-Sensitive Stream Mining

Shaolei Ren, Nikos Deligiannis, Yiannis Andreopoulos, Mohammad A. Islam, Mihaela van der Schaar

***Abstract*—Numerous stream mining applications, such as visual detection, online patient monitoring, video search and retrieval, are emerging on both mobile and high-performance computing systems. These applications are subject to responsiveness (i.e., delay) constraints for user interactivity and, at the same time, must be optimized for energy efficiency. The increasingly heterogeneous power-versus-performance profile of modern hardware presents new opportunities for energy saving as well as challenges. For example, employing low-performance processing nodes can save energy but may violate delay requirements, whereas employing high-performance processing nodes can deliver a fast response but may unnecessarily waste energy. Existing scheduling algorithms balance energy versus processing throughput assuming constant processing and power requirements throughout the execution of a stream mining task and without exploiting hardware heterogeneity. In this paper, we propose a novel framework for *Dynamic Scheduling for Energy* minimization (DSE) that leverages this emerging hardware heterogeneity. By optimally determining the processing speeds for hardware executing classifiers, DSE minimizes the average energy consumption while satisfying an average delay constraint. To assess the performance of DSE, we build a face detection application based on the Viola-Jones classifier chain and conduct experimental studies via heterogeneous processor system emulation. The results show that, under the same delay requirement, DSE reduces the average energy consumption by up to 50% in comparison to conventional scheduling that does not exploit hardware heterogeneity. We also demonstrate that DSE is robust against processing node switching overhead and model inaccuracy.**

***Index Terms*—Energy efficiency, delay-sensitive, scheduling, stream mining.**

I. Introduction

Extracting knowledge and recognizing patterns from continuous and rapid data streams has become ubiquitous today, thereby enabling the emergence of a plethora of real-time stream mining applications such as traffic analysis, financial fraud prevention, disaster information management, video surveillance, and online patient monitoring [5], [6]. Stream mining systems rely on sophisticated machine learning techniques and can be conceptually viewed as processing pipelines that identify data of interest by progressively testing it on classifiers. The classifiers, which are essential elements that partition data into multiple classes and filter out irrelevant information [5], can be deployed in heterogeneous processing hardware available on a high-performance cloud-computing infrastructure or a networked computing cluster of embedded processors [2].

The evolution of stream mining applications has manifested two trends. **(1)** As the number and types of stream mining applications are rapidly increasing, it becomes essential to devise energy-efficient solutions and systems for supporting these applications [1]. **(2)** Many stream mining applications are subject to stringent delay requirements, since their users want responses to their queries quickly and delayed responses are less valuable or may even result in serious consequences (e.g., as in disaster information management or online patient monitoring systems). Moreover, another growing trend is that these applications are increasingly deployed over embedded systems (e.g., real-time motion detection using mobile devices in assistive environments) [3] or large computing clusters (which process interactive data analytics [20]), for both of which energy efficiency is a critical concern. While enormous efforts have been dedicated to improving data mining algorithms (see [30] for an overview), there is currently a limited volume of work on the minimization of energy consumption of delay-sensitive stream mining applications. This is a critical aspect that can become a major hindrance to the wider adoption of stream mining applications over *mobile* or *large heterogeneous computing clusters*.

In this paper, we aim at optimizing energy efficiency of stream mining applications employing cascade classifier chains. Prior studies show that load shedding [8] and classifier topology configuration [5] can be applied to effectively reduce the complexity of stream mining applications, potentially saving the resource consumption (including energy consumption) as a by-product of the aforementioned solutions. Here, we adopt a different and equally important approach: *optimally choosing processing speeds for the hardware executing the classifiers*. Our approach leverages the growing heterogeneity of modern hardware and enables dynamic scheduling of stream mining workloads over a set of heterogeneous processing nodes. Notable examples include heterogeneous multicore processor (e.g., ARM big.LITTLE [16]) and heterogeneous server clusters in which different servers have different power-performance profiles [32]. While our goal is clear, a key issue that remains unsolved spurs our research: selecting processing nodes with low speeds can save energy but may violate the delay requirement, whereas selecting processing nodes with fast speeds can deliver a fast response but may unnecessarily waste energy. Moreover, some classifiers may be more energy-consuming than others, thereby further complicating the opti-

†This work is supported in part by National Science Foundation under Grant No. 1016081. Part of this work was done while S. Ren was at UCLA. S. Ren and M. A. Islam are with School of Computing Information Sciences, Florida International University (sren@cs.fiu.edu, misla012@fiu.edu). M. van der Schaar is with Electrical Engineering Department, University of California, Los Angeles (mihaela@ee.ucla.edu) N. Deligiannis and Y. Andreopoulos are with Electronic and Electrical Engineering Department, University College London (n.deligiannis@ucl.ac.uk, i.andreopoulos@ucl.ac.uk).

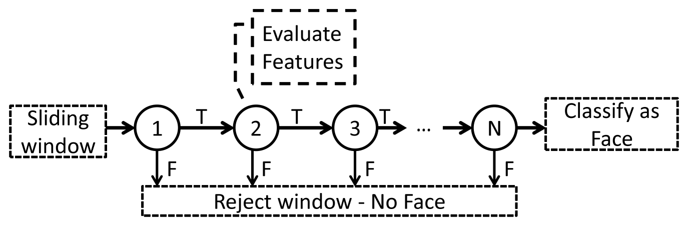

Fig. 1. Cascaded classifier architecture of Viola-Jones face detection [4].

mal choice of processing nodes for classifiers.

Our study aims at addressing energy efficiency of delay-sensitive stream mining applications from a scheduling perspective, which is becoming increasingly important in both mobile systems and large computing clusters. While there are many different scheduling algorithms in other contexts (e.g., dynamic voltage and frequency scaling (DVFS) [12], [13] and scheduling in heterogeneous multicore processors [14], [21]), these studies assume that a stream mining task exhibits constant performance (i.e., constant processing cycles and power consumption) throughout its execution, which does not necessarily hold (as shown in our experiment). Conversely to these state-of-the-art works, we propose a novel algorithm, called *Dynamic Scheduling for Energy* (DSE), which dynamically chooses processing nodes for executing the classifiers in order to minimize the average energy consumption for stream mining, while simultaneously satisfying an average delay constraint.

To evaluate DSE, we build an image stream mining system based on the well-known Viola-Jones classifier chain [4], which extracts features from image sequences and detects the region(s) where human faces appear in the images. For each of the Viola-Jones classifiers, we profile the execution time requirement, measure its frequency of occurrence, and calculate its energy consumption on a set of heterogeneous servers. When applying DSE for delay-constrained and energy-optimal realization of the Viola-Jones classifier chain on this heterogeneous environment, our results show that the average energy consumption is significantly reduced (by up to 50%) in comparison to the best homogeneous system that only chooses a static speed. Beyond this result, we also conduct extensive sensitivity studies to demonstrate the robustness of DSE in cases where some parameters, such as service demand distribution, may not be completely known.

The rest of this paper is organized as follows. Section II describes the model. In Section III, we develop the DSE algorithm. Simulation results are shown in Section IV and related work is reviewed in Section V. Finally, concluding remarks are offered in Section VI.

II. Stream Mining and Job Model

A. Stream mining system

We provide the modeling details of classifiers and processing nodes. For the ease of presentation, we concentrate on a single (binary) classifier chain, whereas a more complex classification tree model is provided in Section III-C. As a classic example, Fig. 1 illustrates the well-known cascade classifier chain architecture of the Viola-Jones algorithm [4] consisting of multiple stages of filters for face detection in images.

1) Classifier: Ordered according to a certain topology (e.g., classifier chain [4], [5]) depending on the specific application, each classifier is responsible for extracting and classifying features of certain interest. Note that each classifier also corresponds to one "stage" of classification [4]. An input stream data is filtered sequentially along the cascade of classifiers and two possible labels may be produced at each stage: "Positive/True" and "Negative/False". Data labeled as "Positive/True" is forwarded to the next classifier if any, while data labeled as "Negative/False" is dropped. We further note that each classifier may be applied multiple times at one stage before the input data is forwarded to the next stage. For example, for face detection within streams of images, a classifier is typically applied on a small block and then scans throughout the entire image; those blocks labeled as "Negative" will not be scanned by the next-stage classifier [4].

In our model, the classifier chain consists of N stages (i.e., N binary classifiers) indexed by $1, 2, \cdots, N$, respectively. We use w_i to quantify the average complexity (i.e., number of required CPU instructions, or resource demand) of the i-th stage of classification ($1 \leq i \leq N$), during which the i-th classifier may be applied once or multiple times (as in the case of face detection [4]). Each classifier has *a priori* selectivity for the incoming data (i.e., *a priori* probability that the data belongs to the "Positive" group, which is an inherent characteristic of the input data), that can be computed on a training and cross-validation data set. Given a fixed classifier configuration, the *a-priori* selectivity and the likelihood of classification errors will probabilistically determine the number of classifiers that will be activated to process the input data [5]. Mathematically, we denote by $f_i > 0$ the probability that the input data is processed by i classifiers (i.e., the input data goes through the first i classifiers prior to being dropped or completed), for $i = 1, 2, \cdots, N$. Equivalently, $f_1, f_2, \cdots, f_N$ can be interpreted as the probability mass distribution of service demand measured in terms of the number of classifiers. We denote the cumulative distribution function (CDF) by $F_i = \sum_{j=1}^{i} f_j$. Although the actual number of classifiers that a particular input data will go through is unknown in advance, the statistical information (i.e., the values of $f_1, f_2, \cdots, f_N$) is readily available by, e.g., online or offline profiling (e.g., [5] [12]) or based on the *a-priori* selectivity of input streams [5]. In Section IV, we shall show the robustness of DSE in the presence of service demand estimation errors.

2) Processing node: We present a general model for processing nodes that are applicable to both mobile systems and high-performance computing clusters. Specifically, we use a general term "node i" to represent the hardware for executing classifier i in the classification cascade. Depending on the system implementation, a "node" may refer to various types of physical hardware: a physical server in large computing cluster [10] or a single core on a heterogeneous multicore processor [16].

We denote the processing speed of node i for executing classifier i by $x_i \in \mathcal{S}_i$, where $\mathcal{S}_i$ is the set of available discrete processing speeds, measured in terms of instructions per second. For notational convenience, we use the vectorial expression $\mathbf{x} = (x_1, x_2, \cdots, x_N)$ wherever applicable. Thus, the average processing time of stage i is given by w_i/x_i, where w_i is the average number of CPU instructions corresponding to classifier i. For stage i, we relate the processing speed to energy consumption via an energy function $e_i = e_i(x_i)$. Here, we explicitly include the subscript i in the energy function $e_i(x_i)$ to emphasize that the energy consumption may be different even though two different stages i and j are executed using the same physical hardware at the same speed, because each stage may involve different types of classification operations that exhibit different power-performance characteristics [4], [22]. Without loss of generality, we consider that a faster node incurs more energy for a given classification stage, i.e., $e_i(x)$ is increasing in x, since otherwise the problem becomes trivial: a slower node will not be used due to its lower performance and higher energy consumption. We further assume that, for each stage of classification $i = 1, 2, \cdots, N$, the energy consumption $e_i(x)$ is convex in the processing speed x, which has been widely considered and validated extensively by both analytical models and practical measurement studies (e.g., [12], [17]).

We express the average energy consumption of the stream mining system as follows

$$\bar{e}(\mathbf{x}) = \sum_{n=1}^{N} \left[\sum_{j=1}^{n} e_j(x_j) \right] \cdot f_n = \sum_{n=1}^{N} [1 - F_{n-1}] \cdot e_n(x_n), \quad (1)$$

which can be briefly explained as follows. The term "$\sum_{j=1}^{n} e_j(x_j)$" represents the energy consumption of an input data that is processed by the first n classifiers (with a probability of f_n), and hence we have the average energy consumption as $\sum_{n=1}^{N} \left[\sum_{j=1}^{n} e_j(x_j) \right] \cdot f_n$. Equivalently, we can rewrite the average energy consumption as $\sum_{n=1}^{N} [1 - F_{n-1}] \cdot e_n(x_n)$, where $1 - F_{n-1}$ is the probability that classifier n is activated.

Remark: In this paper, we focus on optimizing the nodes' processing speeds per classifier stage, and do not consider dynamically adjusting the speeds of classification *within* a single classifier stage. Moreover, in the model, we implicitly assume co-located processing nodes such that the communication delay incurred by migrating a job from one physical node to another is negligible compared to the delay requirement. We shall discuss in Section III-C how to incorporate migration overheads into our model when this assumption does not hold.

B. Job model

Each stream mining service request is referred to as a *job*. As discussed in the previous subsection, a job may be labeled as "negative" and hence discarded by classifier i with probability f_i. Thus, conceptually, a job has a service demand of $\sum_{j=1}^{i} w_j$ with probability f_i. As in [5], we concentrate on the processing delay. In other words, there is at most one job at any time instant such that there is no queueing delay. In practice, this model captures a lightly-loaded system, a periodic system (e.g., video stream mining system) where the jobs arrive periodically with a sufficiently large period, and/or the scenario in which a stream mining request is submitted or admitted only upon the completion of an existing request. The single-job model has been adopted by various studies such as such as [12], [13].

We use the average delay, which is a widely-employed performance metric [5], to capture the responsiveness of stream mining. Given the processing speeds $\mathbf{x} = (x_1, x_2, \cdots, x_N)$, the average delay can be expressed as

$$D_{\text{avg}} = \sum_{n=1}^{N} \left[\sum_{j=1}^{n} \frac{w_j}{x_j} \right] \cdot f_n = \sum_{n=1}^{N} [1 - F_{n-1}] \cdot \frac{w_n}{x_n}, \quad (2)$$

which can be interpreted similarly as we did for (1).

III. Minimizing Energy With Multiple Delay Constraints

This section presents the proposed DSE algorithm that chooses the processing speeds to minimize the average consumption subject to the average delay constraint. We first present the problem formulation and then show how to compute efficiently the optimal processing speeds. Then, we extend DSE to address migration overheads and tree-based classification.

A. Problem formulation

We formulate the energy minimization problem as follows:

$$\mathbf{P1}: \quad \min_{\mathbf{x}} \sum_{n=1}^{N} [1 - F_{n-1}] \cdot e_n(x_n) \quad (3)$$

$$s.t., \quad D_{\text{avg}} = \sum_{n=1}^{N} [1 - F_{n-1}] \cdot \frac{w_n}{x_n} \leq \bar{D}, \quad (4)$$

$$x_i \in \mathcal{S}_i = \{s_{i,1}, s_{i,2}, \cdots, s_{i,M}\}, \ \forall\, i = 1, 2 \cdots, N, (5)$$

where M is the number of available speeds. The subscript i in $\mathcal{S}_i = \{s_{i,1}, s_{i,2}, \cdots, s_{i,M}\}$ is to emphasize that the same processing node may exhibit different speeds on different classifiers [22].

The input to **P1** is the service demand distribution $f_1, f_2, \cdots, f_N$, delay constraint, complexity (or resource demand) of each stage and the energy function, while the output is the optimal processing speeds $\mathbf{x} = (x_1, x_2 \cdots, x_N)$ for classifiers. Due to the discrete processing speed constraint, the problem **P1** falls into combinatorial programming, which incurs an exponential complexity. A simple approach to solving **P1** is: (1) by replacing "$x_i \in \mathcal{S}_i$" with $x_i \in [s_{i,\min}, s_{i,\max}]$ and reformulating **P1** as a convex problem, we apply standard convex techniques [33] to solve the relaxed problem, denoted by **P2**; and (2) we round the obtained continuous processing speeds $x_i^* \in [s_{i,\min}, s_{i,\max}]$ to the closest value in $\mathcal{S}_i$ that is no greater than x_i^*. While this approach automatically satisfies the delay constraint, the energy consumption may be far from the minimum [31].

Algorithm 1 Greedy

```
1: Initialize $x_i = s_{i,\min}$, for $i = 1, 2, \cdots, N$
2: while $\mathbf{x} \neq \mathbf{s}_{\max}$ and constraint (4) is not satisfied do
3:     $\Omega \leftarrow \{i \,|\, i = 1, 2, \cdots, N, x_i < s_{i,\max}\}$
4:     $\Delta x_i \leftarrow \arg\min_{x \in \mathcal{S}_i} (x > x_i), \forall i = 1, 2, \cdots, N$
5:     $i = \arg\min_{i \in \Omega} \{e_i(\Delta x_i) - e_i(x_i)\}$
6:     $x_i \leftarrow \Delta x_i$
7: end while
8: return $\mathbf{x}^* = \mathbf{x}$
```

B. Dynamic Scheduling for Energy – DSE

Below, we develop an efficient branch-and-bound algorithm in the following four steps to yield a sub-optimal solution.

1) Decomposition. We first decompose **P1** into M sub-problems, indexed by $\mathbf{P2}_1, \mathbf{P2}_2, \cdots, \mathbf{P2}_M$. Each sub-problem $\mathbf{P2}_m$ is expressed as follows:

$$\mathbf{P2}_m: \quad \min_{\mathbf{x}} \sum_{n=1}^{N} [1 - F_{n-1}] \cdot e_n(x_n) \tag{6}$$

$$s.t., \quad D_{\text{avg}} = \sum_{n=1}^{N} [1 - F_{n-1}] \cdot \frac{w_n}{x_n} \leq \bar{D}, \tag{7}$$

$$x_i \in \mathcal{S}_i, \; \forall\, i = 2, 3 \cdots, N, \tag{8}$$

$$x_1 = s_{1,m}, \tag{9}$$

where we fix $x_1 = s_{1,m}$ as the m-th supported processing speed for classifier 1 and minimize the average energy $\mathbf{x} \backslash \{x_1\}$.[1] If we can solve all the M sub-problems and select one sub-problem (say, $\mathbf{P2}_m$) that yields the minimum energy, then, combined with $x_1 = s_{1,m}$, we obtain the optimal processing speeds $\mathbf{x}^*$. Each sub-problem itself is an combinatorial problem and can be further decomposed into multiple smaller problems by fixing the processing speed for another classifier. Therefore, the original problem can be solved recursively, which serves as the basis for applying the branch-and-bound technique.

2) Lower and upper bounds. By relaxing the processing speed constraint "$x_i \in \mathcal{S}_i$" with "$x_i \in [s_{i,\min}, s_{i,\max}]$" and solving the relaxed problem **P2** using convex optimization [33], the resulting energy consumption is obtained over a relaxed constraint. This is, therefore, a lower bound on that of the original problem **P1**.

To find an upper bound on the minimum energy in **P1**, we propose a greedy algorithm (see Algorithm 1), which never outperforms the optimal solution to **P1** in terms of the energy consumption. In the greedy algorithm, all the processing speeds are initially set to their minimum values (i.e., $x_i = s_{i,\min}$, for $i = 1, 2, \cdot \cdots, N$). If the average delay constraint is not satisfied, we greedily increase the processing speed such that the total energy increase is minimum (i.e., Line 3–6 in Algorithm 2). Repeat this process until all the processing speeds increase to the maximum or the average delay constraint is satisfied.

Next, we define the following notations that facilitate the algorithm description: $LB(\mathcal{X})$ is the minimum energy obtained by solving **P2** with the constraint $\mathcal{X}$ as its additional input;

[1]We can also fix the processing speed for any classifier other than 1.

Algorithm 2 DSE

```
1: Initialize: $i \leftarrow 0$, $\mathcal{X}_0 \leftarrow \varnothing$, set of leaves of a single-node tree
   $\mathcal{S} \leftarrow \mathcal{X}_0$
2: Compute lower and upper bounds: $L_0 = LB(\mathcal{X}_0)$ and $U_0 = UB(\mathcal{X}_0)$
3: while $U_i - L_i > \epsilon$ or $i < IterateMax$ do
4:     Choose the splitting node: $\mathcal{X}^* = \arg\min_{\mathcal{X} \in \mathcal{S}} LB(\mathcal{X})$
5:     Choose the processing speed to fix: $n = next(\mathcal{X}^*)$
6:     Generate $M$ new constraint sets:
       $\mathcal{X}_{i+1}^1 = \mathcal{X}^* \cup \{x_n = s_1\}, \cdots, \mathcal{X}_{i+1}^M = \mathcal{X}^* \cup \{x_n = s_M\}$
7:     Update the set of leaves:
       $\mathcal{S} \leftarrow (\mathcal{S} \backslash \{\mathcal{X}^*\}) \cup \{\mathcal{X}_{i+1}^1\} \cup \cdots \cup \{\mathcal{X}_{i+1}^M\}$
8:     Compute upper and lower bounds for $M$ new constraint sets:
       $LB(\mathcal{X}_{i+1}^1), \cdots, LB(\mathcal{X}_{i+1}^M), UB(\mathcal{X}_{i+1}^1), \cdots, UB(\mathcal{X}_{i+1}^M)$
9:     Update global upper and lower bounds:
       $L_{i+1} = \min_{\mathcal{X} \in \mathcal{S}} LB(\mathcal{X})$ and $U_{i+1} = \max_{\mathcal{X} \in \mathcal{S}} UB(\mathcal{X})$
10:    $i \leftarrow i + 1$
11: end while
12: Choose the best constraint set thus far:
    $\bar{\mathcal{X}} = \min_{\mathcal{X} \in \mathcal{S}} UB(\mathcal{X})$
13: return $\mathbf{x}^*$ achieved by the greedy algorithm (i.e., Algorithm 1)
    with $\bar{\mathcal{X}}$ as the constraint
```

$UB(\mathcal{X})$ is the energy obtained by using the proposed greedy algorithm (i.e., Algorithm 1) with the additional constraint $\mathcal{X}$ as its additional input. The definitions can be explained using the following example example: if $\mathcal{X} = \{x_1 = s_{1,m}\}$ where $x_{1,m} \in \mathcal{S}_1$, we compute $LB(\mathcal{X})$ by solving **P2** with the additional constraint of $x_1 = s_{1,m}$. We use $LB(\varnothing)$ and $UB(\varnothing)$ to represent the energy obtained by solving the original problem **P2** and by using the greedy algorithm without additional constraints, respectively.

3) Fixing rule. The branch-and-bound algorithm requires a "fixing" rule, which determines the next decision variable to be fixed. We define the "fixing" rule as follows: $next(\mathcal{X})$ is the classifier index that the proposed greedy algorithm selects next to update the processing speed given the constraint set $\mathcal{X}$ as the input. In essence, we select and fix the processing speed for a classifier which, if increased to the next bigger value out of the supported processing speeds, results in the minimum energy increase.

4) Algorithm. We describe our branch-and-bound algorithm in Algorithm 2. The parameter $IterateMax$ is the maximum number of iterations selected based on the desired accuracy and the problem scale. The algorithm generates a tree, where each node represents a constraint set and all the leaf nodes are stored in the set $\mathcal{S}$. The algorithm begins with an empty constraint set $\mathcal{X}_0 = \varnothing$ as the parent node of the tree. In each iteration, we choose a leaf node and split it into new leaf nodes, each of which represents a new constraint set with an additional processing time x_i fixed to be one of the permissible values in $\mathcal{S}_i$. In the splitting process (i.e., Line 4–7), we split the node that corresponds to the constraint set resulting in the minimum energy (obtained by solving **P2** with an additional constraint specified by the node to be split). Besides the maximum number of iterations, another stopping criterion is the difference between the global upper and lower bounds. Specifically, if $U_t - L_t$ is no greater than a sufficiently small positive number ϵ, it is guaranteed that the solution obtained using the greedy algorithm with an appropriate constraint set is suboptimal. Therefore, by increasing $IterateMax$ and using a sufficiently small positive number ϵ, DSE yields a close-to-optimal solution, while the global optimality can be achieved at the expense of increasing the computation complexity. The analysis of convergence rate is beyond the scope of our paper, and interested readers are referred to [31].

C. Extension

In this subsection, we provide a discussion on how to extend the preceding analysis.

1) Migration overheads: Migrating a stream mining job from one processing node to another (e.g., from one core to another in a multi-core mobile device) may incur context switch and hence an overhead: a certain amount of time is wasted during which no nodes can process the job, although the overhead is typically small compared to the delay constraint, as confirmed by several prior studies [12], [21]. Here, we briefly describe how to address the migration overhead. If classifier n and classifier $(n+1)$ are processed on different nodes, let τ_n^o be the migration overhead of migrating a job, for $n = 1, 2, \cdots, N-1$, quantified in terms of the wasted time. In the worst case where a job migrates $(n-1)$ times, the average delay constraint becomes

$$\sum_{n=1}^{N} [1 - F_{n-1}] \cdot \frac{w_n}{x_n} + \sum_{i=1}^{N-1} \tau_i^o \leq \bar{D}. \tag{10}$$

Thus, we can reformulate the energy minimization problem by replacing the delay constraint (4) with (10) to conservatively account for the migration overheads.

2) Classification tree: Section II models a cascade classifier chain for single-concept detection. Now, we extend our model to a classification tree. We follow the same notations as in Section II and still index the classifiers from $i = 1, 2, \cdots, N$, although they may not follow a single chain. We consider that the classification tree consists of L different class labels indexed by $l = 1, 2, \cdots, L$. There is a unique path from the data entry point to each label l (i.e., from the parent node of the tree to each leaf node). We denote by $\mathcal{C}_l$ the set of classifiers along the path from the data entry point to label l, and by f_l the probability that an input data is classified into label l. Thus, if an input data is classified into label l, the energy consumption is $\sum_{i \in \mathcal{C}_l} e_i(x_i)$ and the processing delay is $\sum_{i \in \mathcal{C}_l} \frac{w_i}{x_i}$, where x_i is the processing speed for the node executing classifier i. Therefore, we can formulate the energy minimization problem for a classification tree as

$$\textbf{P3}: \quad \min_{\mathbf{x}} \sum_{l=1}^{L} \left[\sum_{i \in \mathcal{C}_l} e_i(x_i) \right] f_l \tag{11}$$

$$s.t., \quad \sum_{l=1}^{L} \left[\sum_{i \in \mathcal{C}_l} \frac{w_i}{x_i} \right] \cdot f_l \leq \bar{D}, \tag{12}$$

$$x_i \in \mathcal{S}_i = \{s_{i,1}, \cdots, s_{i,M}\}, \forall\, i = 1, \cdots, N, \tag{13}$$

which can be solved using the same approach as **P1**.

(a) (b)

Fig. 2. Examples images from face detection database [29].

IV. Performance Evaluation

In this section, we validate our analysis by performing experiments using a real-world stream mining application. We first describe our experimental setup that involves face detection based on the well-known Viola-Jones classifier chain [4]. We then show that, by varying the processing speeds using DSE, the average energy consumption can be reduced by up to 50% while satisfying the delay requirement compared to the benchmark that does not dynamically choose processing speeds. We also conduct sensitivity studies and show the robustness of DSE.

A. Experimental setup

We first briefly introduce the stream mining application, then describe our implementation, and finally show the profiling results for the Viola-Jones classifier tree.

1) Application: To assess the performance of DSE, we conduct a real-world stream mining experiment for the application of face detection. Specifically, our experimental setup is built upon the basis of the well-known Viola-Jones algorithm [4], which has been extensively applied for real-time face detection on a variety of devices ranging from mobile terminals [24], [25] to wireless visual sensors [26].

In brief, the Viola-Jones algorithm scans each incoming video frame with a search window of $W \times W$ pixels (with $W = 10$ for our experiments), searching for particular human-face features. If a sufficient amount of such features is successfully detected, the specific window is classified as a face. Instead of operating directly on pixel values, the algorithm relies on simple Haar-like features that resemble Haar basis functions [27]. These features can be computed very efficiently using an intermediate representation of the image, referred to as *integral image* [4]. Constructing features in this fashion results in a large feature set associated with each sliding window. Although each feature is efficiently computed, the manipulation and evaluation of such an over-complete set is prohibitively expensive. Hence, the feature set is restricted to a small number of critical features that are effectively trained. Selection and training are performed offline via the use of the boosting machine learning algorithm AdaBoost [28].

In order to boost the classification performance while constraining the computational overhead, Viola and Jones employed a cascade classifier architecture [4] consisting of multiple stages of filters forming a classifier chain, as shown in Fig. 1. According to the cascaded principle, simple classifiers are first tested to reject the majority of sliding windows prior to the execution of more complex classifiers that achieve low false positive rates. Each time the sliding window shifts, the new region within the sliding window will go through the cascade classifier stage-by-stage. Equivalently, the sliding window can first slide throughout the entire image and call respective classifiers for one stage before moving to the next stage; after moving a new stage, classifiers will not be called when the sliding window shifts to those regions labeled as "Negative" by prior stages. While theoretically speaking these two approaches are equivalent, we choose the latter one, because processing speed changes are only possibly needed after the sliding window scans through the entire image and moves to a new classification stage, whereas the former method may incur processing speed changes for each region the sliding window shifts to.

On account of the aforementioned cascaded classification, the Viola-Jones method has a very high accuracy rate. In particular, false negative rates of less than 1% and false positive rates of less than 40%, have been reported, even with a simple filter setup, while the full system benefits from up to $N = 32$ cascaded filters.

2) Implementation: In our C implementation of the Viola-Jones algorithm, we adhere to a multi-scale representation of each video frame (i.e., *image pyramid*). In this way, the face detection can be scale-invariant, thereby allowing for detecting large and small faces with the same sliding window size (being 10×10 pixels in our implementation). We consider a 25-stage ($N = 25$) cascade classifier chain, with each stage comprising multiple Haar filters ranging from 9 to 211. Within a particular stage, the image region will go through these filters in parallel. The output of these filters will be summed up and compared against a per-determined threshold, based on which the decision of the detection is made.

We run our implementation of the Viola-Jones algorithm over the images of the well-known Face Detection Database (FDDB) [29]. The database contains annotations for 5171 faces in a set of 2845 images taken from the Faces in the Wild data set. Two example images in the database are depicted in Fig. 2. Executables written in C and compiled with the gcc compiler (-O3, full optimization) are run on: two different Intel Xeon cores of the Intel 31xx and 56xx families, one AMD processor (Athlon II P320), and the IMGTEC minimorph board (Meta HTP221 GP processor). The Intel and AMD processors comprise common choices for multicore high-performance and desktop architectures today, while IMGTEC minimorph boards can form a low-energy cloud-computing cluster when interconnected via their embedded WiFi mesh connectivity. In our study, the term "processing node" refers to one of the aforementioned Intel, AMD and IMGTEC cores.

Our experiment environment is based on the CentOS linux operating system (and the default Linux OS of the minimorph boards). Given that there is a lack of a commercially-available heterogeneous system that includes all these cores in a single computing environment, following the commonly-used approach in the literature (e.g., [21]), we emulate a heterogeneous processor by: *(i)* repeatedly generating input streams from our data set and classifying them; *(ii)* measuring the processing time (delay) and energy consumption on each processor; and *(iii)* substituting the measured delays and energy consumption with projected delays and energy consumption, as though the input streams are processed by a heterogeneous system that we desire to have in our experiment.

3) Profiling result: The frequency (i.e., the number of times each classifier is accessed) and the average execution time per stage are collected. Energy consumption estimates per classification stage are derived from the measured execution time and real-time power estimates. These estimates are obtained by retrieving real-time operating frequency information from the Linux Hardware Abstraction Layer (HAL) and the corresponding power values provided by the manufacturers.

Table I reports the results for each one of the 25 classification stages in the cascaded architecture illustrated in Fig. 1, averaged over all images in the database and executed by the four aforementioned processors (Intel, AMD and IMGTEC). The classifier frequency results shown in Table 1 corroborate the principle of the Viola-Jones method. Specifically, in initial stages, classifiers are called more frequently, while classifiers lying deep in the cascade are executed fewer times. In addition, one may observe that later-stage classifiers are more complex (i.e., they involve higher execution times per invocation) as they evaluate more Haar-like filters (i.e., up to 211 filters) than earlier-stage classifiers. It is also worth observing that the measurements of different stages do not scale exactly proportionally across different machines. This is due to the variability of the processing performed by each classification stage (i.e., different numbers and types of Haar filters per stage).

Now, we discuss how to apply our model to the Viola-Jones face detection algorithm. Table I provides the following information: the energy consumption and processing time for each classification stage on each different processing node. Hence, the energy function $e_i(x)$ is available (for the given processing nodes), while the (normalized) processing speed $x_i \in \mathcal{S}_i = \{s_{i,1}, \cdots, s_{i,M}\}$ can be readily obtained using the inverse of execution time for each classification stage i. To apply DSE, we also need the service demand distribution information f_i (i.e., the probability that a job is completed after passing through i classification stages), which can be obtained from the "Frequency" column of Table I. As can be seen, the number of accessed classifiers varies significantly across different stages. This is because, during the execution, the sliding window will filter out those regions with little chance of having faces and, hence, classifiers are activated less frequently in later stages. Here, we focus on the energy consumption per classification over the entire image, rather than each individual region covered by one sliding window.

TABLE I

AVERAGE FREQUENCY, EXECUTION TIME AND ENERGY CONSUMPTION PER STAGE IN THE VIOLA-JONES CASCADE OF FIG. 1, EXECUTED ON DIFFERENT CORES.

		Intel Xeon E31225@3.2GHz		Intel Xeon X5690@2.0GHz		AMD Athlon II P320@800MHz		IMGTEC MetaHTP221GP	
Stage	Frequency	Time (ms)	Energy (J)	Time (ms)	Energy (J)	Time (ms)	Energy (J)	Time (ms)	Energy (J)
1	456556	158.140	37.954	214.736	32.762	384.728	26.931	681.131	17.028
2	44823	23.389	5.613	30.918	4.717	53.786	3.765	82.425	2.061
3	12222	9.545	2.291	12.735	1.943	22.013	1.541	29.135	0.728
4	3880	3.634	0.872	4.822	0.736	8.308	0.582	10.542	0.264
5	2004	2.871	0.689	3.794	0.579	6.507	0.455	7.459	0.186
6	975	1.436	0.345	1.883	0.287	3.214	0.225	3.670	0.092
7	531	0.932	0.224	1.245	0.190	2.122	0.149	2.328	0.058
8	352	0.669	0.160	0.903	0.138	1.541	0.108	1.643	0.041
9	273	0.614	0.147	0.817	0.125	1.400	0.098	1.465	0.037
10	204	0.501	0.120	0.673	0.103	1.147	0.080	1.173	0.029
11	146	0.385	0.092	0.520	0.079	0.898	0.063	0.907	0.023
12	122	0.370	0.089	0.495	0.075	0.869	0.061	0.853	0.021
13	102	0.343	0.082	0.456	0.070	0.801	0.056	0.783	0.020
14	91	0.327	0.079	0.437	0.067	0.771	0.054	0.750	0.019
15	84	0.309	0.074	0.411	0.063	0.727	0.051	0.702	0.018
16	79	0.281	0.068	0.373	0.057	0.663	0.046	0.636	0.016
17	73	0.318	0.076	0.422	0.064	0.745	0.052	0.705	0.018
18	69	0.293	0.070	0.388	0.059	0.688	0.048	0.653	0.016
19	66	0.291	0.070	0.388	0.059	0.698	0.049	0.653	0.016
20	59	0.307	0.074	0.408	0.062	0.735	0.051	0.677	0.017
21	58	0.310	0.074	0.409	0.062	0.728	0.051	0.683	0.017
22	56	0.270	0.065	0.358	0.055	0.649	0.045	0.600	0.015
23	52	0.280	0.067	0.374	0.057	0.661	0.046	0.622	0.016
24	49	0.274	0.066	0.367	0.056	0.649	0.045	0.607	0.015
25	47	0.251	0.060	0.335	0.051	0.596	0.042	0.552	0.014

B. Experimental Results

This section evaluates the performance of DSE[2] and demonstrates its effectiveness in reducing the average energy consumption in comparison with a benchmark, i.e., FIX. We also demonstrate the robustness of DSE when switching overheads exist and certain modeling parameters are not accurate. Unless otherwise stated, all the average values are per image. Before showing the results, we describe the benchmark algorithm as follows.

FIX: FIX chooses a fixed processing node to minimize the energy consumption subject to an average delay requirement without dynamically varying the processing nodes across different classification stages.

We choose FIX as our benchmark because it corresponds to the optimal scheduling algorithms over heterogeneous multicore processors subject to average delay requirement [14], [21] that assume a core/speed mode to exhibit a constant performance (i.e., constant processing cycles and power consumption) throughout the execution of the classifier chain within an input (i.e., an image or a set of images). Finally, we remark that our comparisons do not include speed changing within a classification stage (e.g., using DVFS [12], [13]), because such approach can be applied under *both* the proposed DSE and FIX within each classification stage. However, doing so requires the probability distribution of execution time of each classification stage [12], which will further increase the implementation complexity without further highlighting the advantage of dynamic scheduling using DSE over FIX.

[2]As classifiers are much less frequently accessed in the last few classification stages, we do not always allow varying processing nodes stage by stage. In this study, DSE is only allowed to change processing nodes after stages 1,2,3,4,5,11 (i.e., we group stages 6–11 and stages 12–25 into two combined stages).

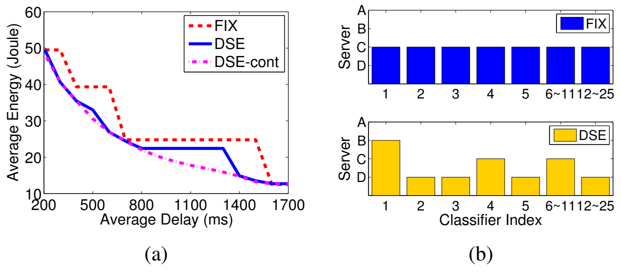

Fig. 3. (a) Energy consumption versus average delay. (b) Selection of processing speeds over classification stages subject to 500ms delay constraint. A: Intel Xeon E31225. B: Intel Xeon X5690. C: AMD Athlon II P320. D: IMGTEC MetaHTP221GP.

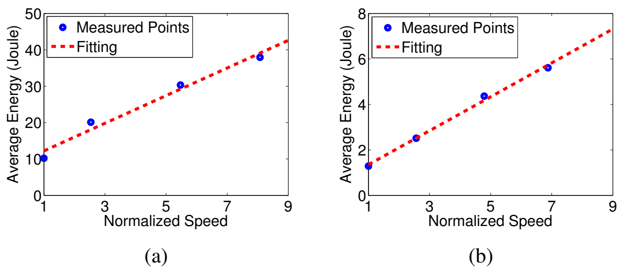

Fig. 4. Energy function. (a) Stage 1: $e_1(x) = 3.80 * x + 8.43$. (b) Stage 2: $e_2(x) = 0.75 * x + 0.61$.

1) Comparison between DSE and FIX: To begin with, we show the comparison between DSE and FIX in terms of energy consumption given various delay constraints. Fig. 3(a) shows that DSE achieves up to 50% energy saving compared to FIX by dynamically choosing the processing nodes across classification stages. This is because, to meet a certain delay constraint, a fixed processing node over the entire course of classifications may be faster than needed and hence unnecessarily waste energy. On the other hand, DSE can dynamically select processing nodes that are just fast enough to meet delay constraints without wasting more energy. Fig. 3(b) illustrates the choices of processing nodes over classification stages subject to an average delay constraint of 500ms. While FIX always chooses the medium-speed processing node (i.e., AMD Athlon II P320800MHz), DSE can dynamically trade processing time for energy yet still satisfy the desired delay constraint. Note that, when the delay constraint is very stringent, the fastest processing nodes have to be used throughout all classification stages, and certainly DSE and FIX are the same in this case due to limited choices of processing nodes. Similarly, DSE and FIX are also equivalent when the delay constraint is very loose.

As a reference, Fig. 3(a) also shows the minimum energy consumption by DSE under processing speeds that can be continuously chosen between the fastest and slowest cores in Table I. The new curve is labelled as DSE-cont. Due to the limited number of measurements, we extend the domain of measured energy function $e_i(x)$ to the interval $x_i \in [s_{i,\min}, s_{i,\max}]$ using data fitting based on mean square errors. Fig. 4 illustrates the fitted energy function for the second and third classification stages, where we define the processing speeds with respect to the slowest processing node. With continuous processing speeds (but between the fastest and slowest cores available in our study), the energy consumption by DSE is clearly a lower bound on any attainable value in practical systems. With more processing nodes available, we expect that DSE will achieve an energy consumption closer to the lower bound.

2) Switching overhead: We now study the impact of switching/migration overheads (e.g., context switch) incurred when processing nodes are changed during classification. Here, we model the switching overhead as a certain amount of time that is wasted during which no processing nodes can process the job. While switching overheads cannot be possibly completely eliminated, they are not major issues for "co-located" processing nodes (e.g., sub-milliseconds for multicore processor in an indexing server of search engine [21], a few milliseconds when switching cores for ARM big.LITTLE architecture on a mobile system [14]). When switching overheads are considered, the energy consumption of DSE will certainly increase. Nonetheless, Fig. 5(a) demonstrates that with a switching overhead of 10ms, DSE still outperforms FIX in terms of energy consumption subject to various delay constraints. To further strengthen this point, we fix the average delay constraint as 600ms and show the energy comparison in Fig. 5(b) with various switching overheads. We see that DSE yields energy savings compared to FIX under the same delay constraint for switching overheads of up to 50ms, which is fairly large in practical systems [14]. For practical values (e.g., in the order of a few milliseconds for heterogeneous multicore processors [14], [21]), DSE is fairly robust against switching overheads.

3) Model inaccuracy: In practice, it may not be possible to perfectly obtain the modeling parameters (e.g., processing speed, service demand distribution, etc.). Here, we focus on imperfect estimation of frequencies that classifiers are accessed in each classification stage. This is a major source of model inaccuracy for face detection applications, because many factors, such as sizes of future images and how many faces they have, will affect the values of frequencies. Equivalently, we can view the frequencies as a reflection of the probability that classifiers are accessed for each sliding window. Moreover, inaccurate frequencies will also cause inaccurate energy consumptions and processing times per classification stage. We dynamically choose processing nodes using DSE given inaccurate frequencies but still use the actual frequencies when calculating the average energy. Fig. 6(a) shows the actual frequencies of classifiers accessed and their estimates for classification stages 2–5 (by adding 30% random noises). Despite the inaccurate estimate of frequencies, Fig. 6(b) shows that DSE still saves energy compared to FIX under various delay constraints. Moreover, it shows that even with inaccurate frequencies, the energy consumption of DSE is fairly close to that with perfect knowledge of frequencies, demonstrating the robustness of DSE against model inaccuracy.

We also conduct other sensitivity studies such as inaccurate energy-performance profiles. The results are similar and hence omitted for brevity.

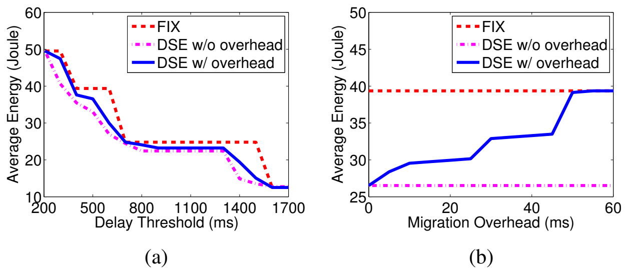

Fig. 5. Impact of switching overhead. (a) Energy consumption under various delay constraints with a switching overhead of 10ms. (b) Energy consumption under various switching overheads with an average delay constraint of 600ms.

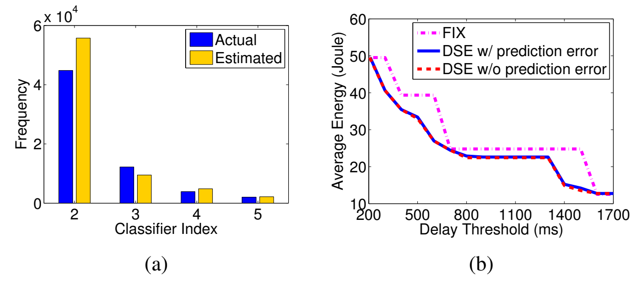

Fig. 6. Impact of model inaccuracy. (a) Actual and estimated frequencies of classifier access in stages 2–5 (b) Energy consumption versus delay constraints.

V. Related Works

Our research lies at the intersection of real-time stream mining optimization and energy-efficient computer systems. We review the related works in the following aspects.

• Stream mining optimization: Improving data mining algorithms and optimally configuring the classifier (e.g., topology, operation point selection) are both effective approaches to improving the stream mining performance [5]. Another important line of research on stream mining optimization relies on processing resource (e.g., CPU, memory, etc.) allocation and load shedding but often without considering responsiveness requirement (see [7]– [9] and references therein).

- *Energy-efficient computing*: Recently, heterogeneous hardware systems, such as heterogeneous multicore processors [16] and heterogeneous clusters [10], have been shown as an appealing architecture for energy saving. In such a system, high-performance but energy-consuming components are combined with low-performance but energy-efficient ones, and a key technique to realize the benefit of energy efficiency is of "job mapping". For example, with predicted service demand, incoming jobs are scheduled to the most "appropriate" core [15], [16], and with known job characteristics, complementary job allocation can be applied to maximize the serve utilization while avoiding resource bottlenecks (e.g., memory-intensive jobs and CPU-intensive jobs are allocated to the same server [11], and conversely, I/O-bound jobs are allocated to energy-efficient servers while CPU-bound jobs are allocated to high-performance servers [10]). Nevertheless, these mapping techniques do not address the delay requirement that is important for stream mining [5], and they are essentially equivalent to FIX without dynamically varying the processing nodes (i.e., job migration) during a job's execution. Prior studies have exploited dynamic migration for energy saving subject to maximum delay constraint in different contexts such as DVFS [12], [13] and heterogeneous multicore processor [21]. However, a critical assumption in these studies is that all the phases of a job (i.e., classification stages in our study) have the same energy function; in contrast, different classifiers have different characteristics in terms of CPU/memory demand, exhibiting diverse energy functions as shown in Fig. 4. Moreover, these studies [12], [13], [21] cannot be applied to classification trees.
- *Others*: Considering large-scale MapReduce computing, [20] proposes to reserve a portion of the servers for processing delay-sensitive jobs, leaving the remaining servers to run at low power states for processing delay-insensitive jobs to achieve energy efficiency. [18] exploits the inherent parallelism and proposes to apply MapReduce on a multicore server to speed up machine learning algorithms. This work has been extended in various directions (e.g., [19] exploits a CPU-GPGPU heterogeneous platform to parallel exact inference to speed up the computation), but algorithm speedup is the focus without addressing energy efficiency. Our approach is complementary to all the above approaches in that it can be applied on top of them to achieve energy efficiency.

VI. Conclusion

In this paper, we investigated energy-efficient design for delay-sensitive stream mining systems from a scheduling algorithmic perspective. We propose a novel algorithm, called DSE, to optimally determine the processing speed for each classifier to minimize the average energy consumption while satisfying the average delay constraint. The key intuition of DSE is that processing nodes should be dynamically chosen to provide just enough computing resources for minimizing energy while meeting the delay requirement. We built a face detection system based on Viola-Jones algorithm and conducted extensive simulation studies to validate DSE. The results showed that compared to the static approach that does not exploit hardware heterogeneity, DSE can reduce the average energy consumption by up to 50% given the same delay requirement. We also demonstrated the robustness of DSE in various scenarios.

References

[1] S. Lakshminarasimhan, P. Kumar, W.-K. Liao, A. Choudhary, V. Kumar, and N. F. Samatova, "On the path to sustainable, scalable, and energy-efficient data analytics: Challenges, promises, and future directions," *IGCC*, 2012.

[2] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M.-Q. Dang, and K. Pentikousis, "Energy-efficient cloud computing," *The Computer Journal*, vol. 53, no. 7, pp. 1045-1051, 2010.

[3] S. K. Tasoulis, C. N. Doukas, V. P. Plagianakos, and I. Maglogiannis, "Statistical data mining of streaming motion data for activity and fall recognition in assistive environments," *Neurocomput.*, no. 107, pp. 87-96, May 2013.

[4] P. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137C154, 2004.

[5] R. Ducasse, D. S. Turaga, and M. van der Schaar, "Adaptive topologic optimization for large-scale stream mining,' *IEEE J. Sel. Topics Signal Process.*, vol. 4, no. 3, pp. 620-636, June 2010.

[6] B. Foo and M. van der Schaar, "A distributed approach for optimizing cascaded classifier topologies in real-time stream mining systems," *IEEE Trans. Image Process.*, vol. 19, no. 11, pp. 3035-3048, Nov. 2010.

[7] B. Babcock, S. Babu, R. Motwani, and M. Datar, "Chain: Operator scheduling for memory minimization in data stream systems," *ACM SIGMOD*, 2003.

[8] Y. Chi, H. Wang, and P. S. Yu, "Loadstar: Load shedding in data stream mining," *VLDB*, 2005.

[9] S. Seshadri, V. Kumar, B. Cooper, and L. Li, "A distributed stream query optimization framework through integrated planning and deployment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 10, pp. 1439C1453, Oct. 2009.

[10] N. Yigitbasi, K. Datta, N. Jain, and T. Willke, "Energy efficient scheduling of MapReduce workloads on heterogeneous clusters," *Green Computing Middleware*, 2011.

[11] W. Xiong and A. Kansal, "Energy efficient data intensive distributed computing," *IEEE Data Eng. Bull.*, 2011.

[12] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with PACE," *ACM Sigmetrics*, 2001.

[13] R. Xu, C. Xi, R. Melhem, and D. Moss, "Practical PACE for embedded systems," *EMSOFT*, 2004.

[14] Y. Zhu and V. J. Reddi, "High-performance and energy-efficient mobile web browsing on big/little systems," *HPCA*, 2013.

[15] J. Chen and L. K. John, "Efficient program scheduling for heterogeneous multi-core processors," *DAC*, 2009.

[16] P. Greenhalgh, "Big.little processing with arm cortex.-a15 & cortex-a7," *ARM Whitepaper*, 2011.

[17] A. Gandhi, M. Harchol-Balter, and C. L. R. Das, "Optimal power allocation in server farms," *ACM Sigmetrics*, 2009.

[18] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, "MapReduce for machine learning on multicore," *NIPS*, 2007.

[19] H. Jeon, Y. Xia, and V. K. Prasanna, "Parallel exact inference on a CPU-GPGPU heterogenous system," *ICPP*, 2010.

[20] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz, "Energy efficiency for large-scale MapReduce workloads with significant interactive analysis," *EuroSys*, 2012.

[21] S. Ren, Y. He, S. Elnikety, and K. S. McKinley, "Exploiting processor heterogeneity in interactive services," *USENIX ICAC*, 2013.

[22] R. Kotla, A. Devgan, S. Ghiasi, T. Keller and F. Rawson, "Characterizing the impact of different memory-intensity levels," *IEEE Workshop on Workload Characterization*, 2004.

[23] V. Balakrishnan, S. Boyd, and S. Balemi, "Branch and bound algorithm for computing the minimum stability degree of parameter-dependent linear systems," *Intl. J. of Robust and Nonlinear Control*, vol. 1, no. 4, pp. 295-317, Oct.-Dec. 1991.

[24] F. M. Ciaramello and S. S. Hemami, “Real-time face and hand detection for videoconferencing on a mobile device,” *Workshop on Video Processing and Quality Metrics for Consumer Electronics*, 2009.
[25] J. Ren, N. Kehtarnavaz, and L. Estevez, “Real-time optimization of viola-jones face detection for mobile platforms,” *IEEE Dallas Circuits and Systems Workshop*, 2008.
[26] M. Camilli and R. Kleihorst, “Demo: Mouse sensor networks, the smart camera,” in *ICDSC*, 2011.
[27] C. P. Papageorgiou, M. Oren, and T. Poggio, “A general framework for object detection”, *ICCV*, 1998.
[28] Y. Freund and R. E. Schapire, “A desicion-theoretic generalization of on-line learning and an application to boosting,” *Springer Computational Learning Theory*, pp. 23C37, 1995.
[29] V. Jain and E. Learned-Miller, “Fddb: A benchmark for face detection in unconstrained settings,” *UMass-Amherst Tech. Rep. UM-CS-2010-009*, 2010.
[30] T. Mitchell, *Machine Learning*, New York: McGraw-Hill, 1997.
[31] S. Boyd, A. Ghosh, and A. Magnani, *Branch and Bound Methods*, http://www.stanford.edu/class/ee392o/bb.pdf, 2003.
[32] L. A. Barroso, J. Clidaras, and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, Morgan & Claypool Publishers, 2013.
[33] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Belmont, MA: Athena Scientific, 1989.