A Pareto Framework for Data Analytics on Heterogeneous Systems: Implications for Green Energy Usage and Performance

Aniket Chakrabarti, Srinivasan Parthasarathy, and Christopher Stewart
The Ohio State University
chakrabarti.14@osu.edu,{srini,cstewart}@cse.ohio-state.edu

***Abstract*—Distributed algorithms for data analytics partition their input data across many machines for parallel execution. At scale, it is likely that some machines will perform worse than others because they are slower, power constrained or dependent on undesirable, dirty energy sources. It is challenging to balance analytics workloads across heterogeneous machines because the algorithms are sensitive to statistical skew in data partitions. A skewed partition can slow down the whole workload or degrade the quality of results. Sizing partitions in proportion to each machine's performance may introduce or further exacerbate skew. In this paper, we propose a scheme that controls the statistical distribution of each partition and sizes partitions according to the heterogeneity of the computing environment. We model heterogeneity as a multi-objective optimization, with the objectives being functions for execution time and dirty energy consumption. We use stratification to control skew. Experiments show that our computational heterogeneity-aware (Het-Aware) partitioning strategy speeds up running time by up to 51% over the stratified partitioning scheme baseline. We also have a heterogeneity and energy aware (Het-Energy-Aware) partitioning scheme which is slower than the Het-Aware solution but can lower the dirty energy footprint by up to 26%. For some analytic tasks, there is also a significant qualitative benefit when using such partitioning strategies.**

***Keywords*-Large Scale Analytics Framework; Pareto Frontier;**

I. Introduction

Distributed algorithms for data analytics partition their input data across many machines. The de-facto approach typically involves a simple scheme such as random or round-robin. The size and content of each data partition can significantly affect the performance and quality of analytics. Size matters because some machines within a rack or data center perform worse than others. Slow machines should process small partitions. Typical solution in this space is a work stealing [1] approach (whichever node finishes its share of partitions, randomly selects a partition from another slower node). However, traditional workstealing based solutions will not scale for distributed analytics workloads as these workloads are typically sensitive to the payload (content) along with the size of data. Content matters because statistical skew across partitions can slow down an analytics algorithm and degrade the quality of its results.

Consider frequent pattern mining [2]. For large datasets, the distributed algorithm simply divides the data into partitions and runs frequent pattern mining locally on individual partitions. Consequently, some of the locally frequent patterns are not globally (considering entire data) frequent and hence an additional scan of the generated candidate patterns is required to prune those. Essentially the total number of candidate patterns represents the search space – the more the number of candidate patterns, the slower the run time. These false positive candidate patterns arise due to the statistical skew across data partitions.

An approach [3] for data partitioning in the homogeneous context based on data stratification (grouping similar or related datum into strata) and then leveraging this information for data partitioning was proposed recently. However, a limitation of this approach is that it does not account for the inherent heterogeneity present in modern data centers. Most real-world data centers are increasingly heterogeneous in terms of both their computational capabilities as well as the available clean energy they use. There are many reasons for such *heterogeneity* such as equipment upgrades and power-performance tradeoffs within modern processor families. We describe the types of heterogeneity in section II.

In this work, we present a partitioning scheme for data analytics on heterogeneous clusters. Our approach scans the data before initiating the analytics workload and identifies similar content. It then builds partitions that reflect the global distribution of data and relative processing capacity of each node. Our approach overcomes several challenges. First, we significantly reduce resources needed to find similar content. We use a lightweight data sketching approach that works in one pass using only in-memory computation. Second, we profile each machine's processing capacity by using small, samples to benchmark the target analytics workload. These samples should be representative of the partitions on which the actual algorithm will run. And then we learn the execution time objective function using progressive sampling, where the samples are representative of the partitions. We take samples of increasing size and run the actual algorithm on them to measure time. We then fit a function to predict execution time given the input data size. The dirty energy consumed by each partition depends on the execution time on each partition and the amount of renewable energy available to the physical server hosting that partition. To model the renewable energy availability we use the PVWATTS simulator [4] from NREL. Combining this with the aforementioned execution time func-

tion, we get our objective function for predicting the total dirty energy consumed by a job. Our goal is to minimize the total dirty energy consumption as well as minimize the maximum running time across all partitions. Hence we frame a multi-objective optimization problem, where the objective function is essentially a weighted average of the objective function with respect to execution time and the objective function with respect to dirty energy consumption. The weighing parameter controls the tradeoff between speed and energy consumption. The solution to thatgives the partition size distribution. This results in a well load balanced job execution.

Specifically, the contributions of this paper are as follows:

- We characterize the innate heterogeneity in today's cloud computing environments. We explain why special care is needed while accounting heterogeneity in case of distributed analytics algorithms. We show that to provide time and energy aware solution, we need to model the hardware specifications as well as the underlying data distribution.
- We provide a simple yet principled approach to model the heterogeneity problem in the computing environment. We describe a method to learn the objective function with respect to the execution time by using progressive sampling technique. This method is aware of both the machine capacities and the content of data. We frame a multi-objective optimization problem for time and energy, which can be solved efficiently using linear programming technique.
- We build our partitioning framework as a middleware on top of the popular NoSQL store Redis. We conduct a thorough testing of our framework on three popular data mining workloads on five real-world data sets and found out that we get up to 51% reduction in time while optimizing for time only and we get up to 31% reduction in time and 14% reduction in dirty energy consumption when we optimize them simultaneously.

II. Motivation

Data centers are often heterogeneous in terms of the performance of their constituent systems. This is inevitable since nodes fail periodically and are often replaced with upgraded hardware. Even when nodes have similar processing capacity, the processing rate of individual virtual machines can still vary. For example, cloud instances hosted on Amazon EC2 with the same hardware specifications exhibit 2X variation in throughput [5]. Another type of heterogeneity increasingly common to data centers is the varying dirty energy footprint of different physical servers. Some leading contemporary designs include:

1. [6] proposes to put the grid ties and renewable supplies at rack level or individual server level rather than at the data center level. This allows data centers to concentrate the green energy as much as possible to the users requesting it.

2. iSwitch [7] envisions that in future data centers, different racks will have different power sources, some might be fully powered with green or dirty energy, while some racks might be powered by both and jobs will be placed to minimize the usage from purely grid tied racks (guarantees availability).

3. Another design gaining prominence [8] is the geo-distributed data centers where jobs are scheduled to use servers from different geographical regions to maximize the use of green energy.

Additionally, a key challenge, for a large class of analytic workloads running on such data centers, are their irregular access patterns and their payload (input parameters, data skew) dependency. Simply partitioning the data while accounting for heterogeneity alone will lead to sub-optimal solutions. The reason is, the time taken to process a partition of data is dependent on the statistical distribution of data as well as the capabilities of the node (processing speed, green energy harvesting).

Here we propose a novel framework that partitions data in a payload-aware way such that the total execution time is reduced while simultaneously accounting for the dirty energy footprint of individual servers. Importantly we note that optimizing for energy (at least the way we have described it) is somewhat at odds to optimizing for performance – in other words there is a Pareto-optimal tradeoff to be explored here [9]. To reiterate, we note that for the problem we tackle in this paper, optimizing for energy is not equivalent to optimizing the "race to halt" (or performance) [10].

III. Methodology

The key elements of our methodology are: i) the specific tasks (which in the context of big data applications are irregular and data dependent in nature); ii) the heterogeneous processing capacity within the data-center-of-interest; iii) the heterogeneous energy footprint within the data-center-of-interest; and iv) the inherent data distribution of the payload. Specifically, we seek to estimate task complexity, heterogeneous processing capability, clean energy availability within the data center, and payload characterization, respectively. We propose to estimate these quantities as follows. First, in an effort to simplify the model we couple the first two elements and seek to estimate the task complexity on a specific system (set of individual machines) by operating on a small sample problem. We note that this estimate will vary across different data analytic tasks and across different datasets (a desirable property), but is a one-time cost (small) and will be amortized over multiple runs on the full dataset. Second, for clean energy availability, we estimate green energy availability by relying on a forecasting strategy. Finally, we leverage the idea of data stratification for characterizing the payload distribution and to facilitate a partitioning strategy that accounts for both energy as well as processing heterogeneity.

Using the estimates derived (as above) our proposed solution casts the partitioning problem as a multi-objective optimization. The solution to this problem is then used to automatically devise an appropriate partitioning strategy in the presence of environment heterogeneity. The key components of our partitioning framework (Figure 1) are: i) a task-specific computational heterogeneity estimator; ii) available green energy estimator; iii) data stratifier (for payload characterization); iv) a Pareto-optimal modeler and v) a data partitioner.

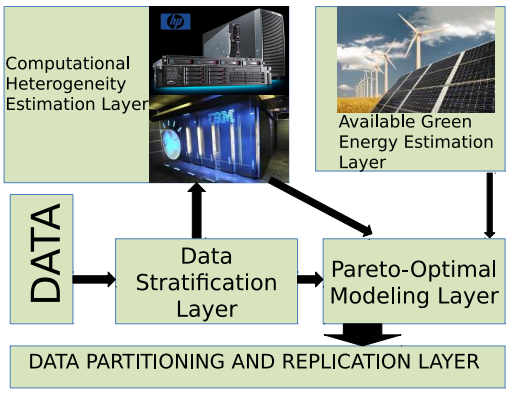


Fig. 1. The entire framework

A. Task-specific Heterogeneity Estimator (I)

As described earlier, in a heterogeneous environment, the time taken by a machine for a specific task depends on the available resources of that machine (speed, time-share, and memory capacities), the underpinning complexity of the task as well as the dataset characteristics (size, statistical distribution of payload). In order to model the task specific heterogeneity, we derive a utility function **f** for execution time that accounts for task complexity and available machine resources. Given a sample input payload, an algorithm and a machine, this function can estimate the execution time of that specific task on that payload in the given machine.

We learn the utility function for time **f** by adapting progressive sampling [11] as follows. We take multiple samples of the input data while increasing sample size from 0.05% up to 2% of the data, and run the actual algorithm on these samples and note the execution time for each run on each node in the system. From these (sample size, execution time) pairs, we fit a linear regression model for predicting runtime of the algorithm on any input size. We discuss this choice in more detail in section III-D. We learn a regression model specific to each node in the cluster. This accounts for the difference in terms of machine speed heterogeneity as we now have execution time models for each machine. This is better than using the stipulated machine CPU speed in three aspects:

1. The CPU speed does not always reflect the true processing rate of an algorithm as there are other factors impacting the speed of an algorithm such as amount of IO required, cache miss pattern, memory access pattern and so on.

2. In virtualized environments, multiple virtual servers are co-located on the same physical host. Two virtual servers with exactly the same configurations could still exhibit different processing rates. One possible reason could be that one of the virtual servers is located on a physical machine which has extremely high load at that instance of time. So the utility function **f** cannot be static, and it has to be learned dynamically.

3. The processing time also depends on the distribution of the data for most data mining algorithms, and the CPU speed cannot capture that aspect.

Our regression model evidently solves problems 1 and 2, as we are running the actual algorithm on the samples, the model learned takes into account the factors such as CPU to IO ratio, cache and memory access patterns and so on. This model can also solve the problem described in the 3rd point (data distribution is also a factor in determining runtime) if we can guarantee that the samples generated during progressive sampling phase are representative of the final data partitions on which the algorithm will run on. We can do this by the stratification process which is described in section III-C.

The total execution time for a particular algorithm on node i will be $f(x_i) = m_i x_i + c_i$, where m_i, c_i are the learned regression coefficients for node i and x_i is the number of data elements on node i.

B. Available Green Energy Estimator (II)

In order to account for the dirty energy footprint across individual machines, we need to predict the amount of renewable energy available to each machine. Hence we need a utility function **GE**, which can predict the amount of renewable energy available to a machine over a time interval. In future, we expect such information will be provided by the data center service provider in terms of carbon ratio guarantee or carbon budget. In the current context, we can model the renewable energy availability in a manner similar to Goiri et al. [12]. According to this model, available renewable energy at hour t is $GE(t) = p(w(t))B(t)$, where $B(t)$ is renewable energy available under ideal sunny conditions, $w(t)$ is the cloud cover and $p(x)$ is attenuation factor. $p(x)$ and $B(t)$ are learned from historical data and $w(t)$ is available from any weather forecast service. To compute availability over an interval, one can sum the GE function over that interval.

Concretely, for the purposes of this work, we leverage the PVWATTS simulator [4] to obtain energy traces for different geographic locations at different points in time. The simulator takes as input the specifications of the solar panel and the location of the solar panel, and based on NREL's weather database and weather models, it outputs the renewable energy production. Though the simulator provides per hour average, one can rescale it to per second average for greater precision. Again we will learn separate models for nodes based on which geographical region the node is from. So for dirty energy footprint of a node for hour t, we can use $g(x_i) = E_i f(x_i) - \sum_{t=1}^{f(x_i)} GE_i(t)$, where E_i is the total energy consumption rate of node i, x_i is the number of data elements in node i and $GE_i(t)$ is the predicted green energy for the hour t for node i.

C. Data stratifier (III)

The job of the stratifier hence is to cluster the input data (payload) into a set of strata where each stratum consists of similar data elements. Such a stratification can then be leveraged by our modeler, in conjunction with estimates on computational heterogeneity and green energy availability, to produce a Pareto-optimal partitioning strategy.

The challenge in the stratification step is to do so efficiently in the presence of complex data elements of varying lengths or dimensions (e.g. documents, transactions) while modeling a range of data types from structured (trees, graphs) to unstructured (text). For this purpose, we first sketch the input data to low dimensional sets using a domain-specific hash function. Then we use a clustering algorithm similar to the Kmodes algorithm to create the strata as outlined previously by Wang et al. [3].

1. Represent the high dimensional data as a set of items. Currently, we support tree, graph and text data. For trees, we first represent them using Prufer sequences [13]. We then extract pivots from the Prufer sequences using the least common ancestor relationship in the tree. For example, a pivot (a, p, q) would mean node a is the least common ancestor of nodes p and q. Each tree in the input data set is represented as a set of pivots. For graph data sets, we use adjacency list as the pivot set (set of neighbors). For text datasets, we represent each document as a set of words in it. The important thing to note is, *at the end of this step, we have converted our input data type to set data, so now operations can be done in a domain independent way.*

2. The aforementioned sets can still be very high dimensional if the input data is high dimensional. The next step is hence to project the high dimensional sets to a low dimensional space (called sketches) and compute similarity in the low dimensional space that can approximate the similarity of the original sets. We use Jaccard coefficient as a measure of similarity between sets. If x and y are the two sets, the Jaccard similarity is given by: $sim(x, y) = \frac{|x \cap y|}{|x \cup y|}$ We use a locality sensitive hash function call min-wise independent permutations [14] by Broder et. al. to generate the sketches and compute approximate Jaccard similarity very efficiently without much loss in accuracy. Let π be a random permutation of the universal set in question, then the min-wise independent permutation hash function on a data point x is as follows: $h_\pi(x) = min(\pi(x))$ Since the cardinality of the universal set can be extremely large, the above hash function can be very expensive to compute, so we use an approximate algorithm called the min-wise independent linear permutations [15] for computing the hash functions. At the end of this step, *we are left with a sketch of the original input data, and the sketch is of orders of magnitude smaller in size that the original input data.* As a result, subsequent operations such as stratification can be done in a very efficient manner. This may be extended to a wide class of similarity measures and the accuracy of the sketch can be controlled by the sketch size [16], [17].

3. Once compact representation in a low dimensional space is done for all the data points, the final step is to cluster on the sketches to create the strata. We use the compositeKModes clustering algorithm proposed by Wang *et al.* [3] to create the clusters. The standard Kmodes algorithm has the following problem. The cardinality of the universal set is very high and since the small sketches contain very few items, chances of every sketch getting matched to a cluster center is very low. Consequently, a large number of data points cannot be assigned to any of the clusters because the data points' sketch set has zero-match with the cluster center's sketch set. To overcome this, we use the compositeKModes algorithm, where instead of a cluster center sketch being the mode of each attribute in the feature space, the center sketch maintains L highest frequency elements for each attribute ($L > 1$). In this case, the probability of zero-match decreases significantly as an attribute element in the data point has to match only one of the L values for the same attribute in a center's attribute list. And this variant of Kmodes can also be shown to hold the convergence guarantees of the original Kmodes algorithm. *Using sketch clustering, the input data can be successfully stratified into clusters.*

D. A Pareto-optimal Model (IV)

Now our goal is to simultaneously minimize the execution time across all partitions, and minimize the sum of the dirty energy consumed by all partitions. Formally, the problem we wish to optimize can be succinctly described as:

$$minimize(v, \sum_{i=1}^{p} g(x_i))$$

$$s.t.\ \forall i,\ v \geq f(x_i), \forall i, x_i \geq 0,\ and\ \sum_{i=1}^{p} x_i = N$$

In the aforementioned formulation v represents the maximum running time across all partitions and $\sum_{i=1}^{p} g(x_i)$ is the total dirty energy consumed by all partitions. Hence the above is a multi-objective optimization problem, with the two objective functions being v and $\sum_{i=1}^{p} g(x_i)$. We wish to find the Pareto frontier [9] of solutions. A solution is a Pareto efficient or optimal one [9] if none of the objectives can be improved upon without degrading at least one. A Pareto frontier is a set of all Pareto efficient solutions. More formally, in terms of our formulation, let v_{xp} and $\sum_{i=1}^{p} g(xp_i)$ be the values of the objective functions at a solution vector $\vec{xp}$. For this solution to be a Pareto optimal one, it must satisfy following condition: for any other solution vector xq, either $v_{xq} \geq v_{xp}$ or $\sum_{i=1}^{p} g(xq_i) \geq \sum_{i=1}^{p} g(xp_i)$.

We solve the multi-objective optimization problem using a technique called scalarization [18]. Here a single objective function is formed by taking the weighted mean of the multiple objectives. Then any single objective optimization technique can be applied. It can be proved that the solution vector we get by the scalarization technique is a Pareto optimal one [18]. By applying scalarization our problem formulation becomes,

$$minimize(\alpha v + (1 - \alpha) \sum_{i=1}^{p} g(x_i))$$

$$s.t.\ 0 \leq \alpha \leq 1, \forall i,\ v \geq f(x_i), \forall i, x_i \geq 0,\ and\ \sum_{i=1}^{p} x_i = N$$

α is the weight factor which controls the tradeoff between the objectives execution time and dirty energy consumption. If

we approximate the value of function GE_i to be the mean renewable energy availability rate over a certain period of time that includes the job execution time, then the above formulation becomes a linear programming problem. If the mean renewable energy availability rate is $\bar{GE}$, then for each node i, the factor $E_i f(x_i) - \sum_{t=1}^{f(x_i)} GE_i(t)$ becomes, $E_i f(x_i) - \bar{GE}_i f(x_i) = k_i f(x_i)$, where k_i is a node specific constant. Then our formulation can be further simplified to:

$$\alpha v + (1-\alpha)\sum_{i=1}^{p} g(x_i) = \alpha v + (1-\alpha)\sum_{i=1}^{p} k_i f_i(x_i)$$
$$= \alpha v + (1-\alpha)\sum_{i=1}^{p} k_i(m_i x_i + c_i)$$

The described formulation is a linear programming problem and hence can be efficiently solved. The solution to this linear programming problem always results in a Pareto-optimal solution, i.e. a change in the solution vector will degrade at least one of the objective functions. Specifically, optimality is guaranteed when the execution time is approximately linearly related to the data size and the fluctuations in the renewable energy availability are minimal so that the availability is close to the mean energy supply. Empirically, even when these conditions are not satisfied, this model will perform better than partitioning naively with equal sized partitions as we shall shortly demonstrate. Though in that case, a better solution will exist.

Another option we considered and empirically evaluated, is to fit a more general functional form to the utility functions, such as higher order polynomials for the regression model. Theoretically, this makes sense as any arbitrary function can be approximated by polynomials using the Taylor approximation. But practically it is not a feasible option as such models will take a very high number of samples to fit the curve properly. Too few points will invariably over fit the points to the model. And we cannot afford too many samples in our progressive sampling step as collecting a sample implies running the actual algorithm on a small sample of the data. Under these circumstances the linear regression model was found to be quite effective on all configurations we evaluated and is moreover easily trained with very few samples and the resulting formulation leads to a linear programming problem, that can be efficiently solved.

Usually, for a multi-objective optimization, there is a set of Pareto optimal solutions. This set is known as the Pareto frontier. Our formulation, which is a weighted mean of the individual objectives, generates only one point on the Pareto frontier. The user-defined parameter α controls which point we get in the Pareto frontier. Hence, setting α to a high value will imply a partitioning scheme where time will be improved more than energy and setting α to a lower value will optimize the energy function better. In fact, our Het-Aware scheme is a special case where α is set to 1.0. The system then will only optimize for time. Note that selecting α can be challenging as the scale of the two objective functions (time and energy) are different. The energy function has a much higher scale than the time function. This implies, to focus more on optimizing time than energy, we have to set a very high value of α (α is the coefficient of the time function). We believe in future this problem can be avoided by normalizing both the objective functions to 0-1 scale, and then both functions will be equally sensitive to changes in α.

E. Data Partitioner (V)

The final component of our framework is the data partitioner. This component is responsible for putting the final data partitions into the machines based on the output of the modeling step. Currently, we support the final partitions to be data partitions stored on disk or data partitions stored on Redis NoSQL store. In future, we plan to use a NoSQL store manager to manage performance and fault tolerance [19]. After the optimization step is done, the framework already knows how many data items to put in each partition. Our data partitioner supports two types of partitions. Both of the schemes are driven by the stratification process.

- **Making each partition representative of payload:** The goal here is to make each partition a representative of the entire data. Such a representative partitioning can be achieved by making each partition a stratified sample without replacement of the data. Cochran [20] showed that a stratified sample approximates the underlying true data distribution much better than a simple random sample. As a result, our partitions will be good representatives of the global data, especially if the data has a large number of strata and each partition is relatively small with respect to the total input data. Since our stratification step already creates the strata, we can proportionally allocate elements to each stratum to create the required partitions.
- **Placing similar elements together:** The goal here is to group similar types of data items together. Again we can achieve such a partitioning scheme by using the strata created by our stratification process. Ideally, we would like to make individual stratum a partition by itself. This would ensure minimum entropy for all partitions. But we have to take to into account the constraint set by the optimizer, it has already decided what the partition sizes are, to optimally load balance. And usually, the number of strata are much higher than the number of partitions. In order to do the partitioning, in this case, we first order the elements one according to the strata id, i.e. all elements of strata 1 followed by elements of strata 2 and so on. Once this ordering is created, we create the partitions by taking chunks of respective partition sizes from this ordered data.

Note that, samples of both kinds of partitions can be generated by the stratifier as only the data clusters are required. Those are the samples which the stratifier feeds to the heterogeneity estimator so that in the progressive sampling step, the samples are representative of the final partition payload. This makes the heterogeneity estimator aware of the payload characteristics.

IV. Implementation

Our data partitioning framework is implemented in C and C++. We use the C library API of Redis NoQSL store as our underlying storage medium. Note that we do not use the cluster mode of Redis as in that mode we do not have control over which key goes to which partition and our whole idea relies on the fact that we will be able to place data items according to our stratification and optimization rules. Hence, we run one instance of Redis server in each of our cluster nodes, and manually manage communication from the framework middleware level. We need a global barrier module for our framework as the pivot extraction, sketch generation, sketch clustering and final data partitioning have to be separated by barriers. We used the atomic *fetch-and-increment* command provided by Redis to create a global barrier routine. Since there could be millions of data items, each of which can be a high dimensional set, storing them could imply millions of get/put requests in Redis and many of those requests could be to remote machines. This can evidently cause a huge performance hit. We use a storage data structure which can avoid this excessive get/put requests on Redis. Instead of storing the individual attribute values of a data item, we store the item as a sequence of raw bytes and we maintain a list of such sequences for a list of data items. The first four bytes in the sequence contain the length of the data object. Note we use the Redis list structure here. This gives us the freedom to access the entire data set of a partition in a single get/put operation, and the access to individual data items from a get/put request as well. To further improve batching of requests we use the pipelining feature of Redis, where requests are batched up to the preset pipeline width and then sent out. In Redis, this is known to substantially improve the response times.

There are three tasks in our framework which are done in a centralized fashion - the global barrier routine, the clustering and creation of the representative data sample which every node will run on to get runtime and energy consumption estimates. Note that we chose to do the clustering in a centralized manner as the compositeKmodes algorithm is run on the sketches rather than the actual data. The size of the sketches of a dataset is of orders of magnitude smaller than the raw data size, which is why it is easy to fit in a single machine. As a result, the clustering can run with zero communication overhead. We saw that doing the clustering in distributed fashion over the sketches is prohibitive in terms of runtime. Even though the two tasks have to be managed by a master node, they need not be the same node. In other words, we choose two separate nodes in the cluster for the two tasks. This gives us some level of decentralization and better load balancing. We also choose type 1 nodes (fastest) as the master nodes if available. If not, then we select type 2, type 3 or type 4 in that order of priority.

V. Experimental evaluation

In this section, we seek to examine the efficacy of the proposed Pareto framework for analytic workloads on heterogeneous systems. Our workloads are drawn from those commonly used in the search, news and social media industry and include both the analysis of structured (e.g. graph) and unstructured (e.g. text) data. Specifically, we use two types of distributed workloads: (i) frequent pattern mining - a compute-intensive workload, where even if the input data size is small, the intermediate data can be huge and (ii) compression - a data-intensive workload that usually runs on huge quantities of data. We seek to answer the following questions:

- Is there a tangible benefit to heterogeneity-aware partitioning for such workloads? For both unstructured and structured data workloads?
- How effective is the Pareto-optimal model? Does using different values of α result in interesting differentials in runtime and dirty energy usage?

A. Setup

We use a cluster of machines to run our experiments. The individual nodes consists of 12 cores with Intel Xeon 2.2GHz frequency. Each machine has a RAM of 48GB. Since the experiments we run are cluster heterogeneity-aware, and the machines in this cluster are homogeneous, we need to introduce heterogeneity both in terms of speed and renewable energy availability. We introduce heterogeneity in the following way:

1. We use 4 different types of machine speeds in our experiments. The idea is to use machines with relative speeds $x, 2x, 3x$ and $4x$. The way we do it is by introducing busy loops in the homogeneous cluster. Since there are 12 cores per machine, type 1 nodes have no busy loops, type 2 nodes will have 12 busy loops, type 3 nodes will have 24 busy loops and type 4 nodes will have 36 busy loops running in parallel.

2. We use the NREL simulator [4] to simulate renewable energy heterogeneity. Again we introduce 4 types of nodes. We select 4 of Google's data center locations and create renewable energy traces for those locations from the weather database and models of the PVWATTS simulator. We found out the server power consumption of each machine from HP SL server specifications (1200 WATTS 12 cores). We used the individual processor power consumption value from Intel Xeon (95 Watts), which implies base operating power is (1200 - 95*12) = 60 Watts. We then generated the 4 types of machines by running 0, 12, 24, 36 busy while loops in the 4 machines as described above. And we assumed that the fastest machine has 4 cores, 2nd fastest 3 cores, then 2 cores and the slowest one with 1 core. Therefore, the power consumption of the 4 types of machines is $60 + 4 * 95 = 440$ Watts, $60 + 3 * 95 = 345$ Watts, $60 + 2 * 95 = 250$ Watts and $60 + 1 * 95 = 155$ Watts respectively.

B. Datasets

We use 5 real-world datasets from 3 different domains namely - graphs, trees and text datasets. Our datasets vary from 50,000 trees set to 15 million nodes graph and since they are collected from different domains, the underlying data distributions and characteristics will largely vary. Table I

Dataset	Type	Size
SwissProt	Tree	# of trees - 59545, Nodes - 2977031
Treebank	Tree	# of trees - 56479, Nodes - 2437666
UK	Graph	Nodes - 11081977, Edges - 287005814
Arabic	Graph	Nodes - 15957985, Edges - 633195804
RCV1	Text	# of docs - 804414, vocab size - 47236

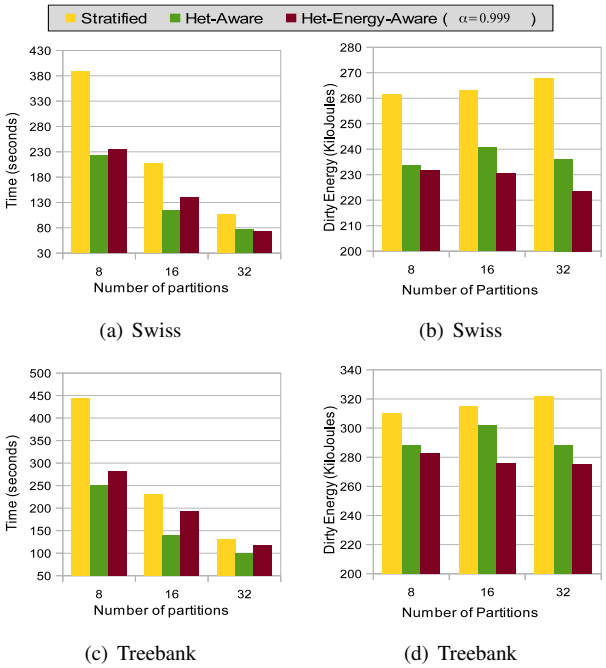
TABLE I. DATASETS

contains the description of the datasets. The datasets are collected from [21], [22], [23].

C. Applications and Results

1) Frequent Pattern Mining: Frequent pattern mining has been one of the most common data mining applications. The goal of the frequent pattern mining problem is to find the frequent co-occurring items in the entire data set. The co-occurring items have to be present in at least a certain percentage (called support) of the entire dataset. There has been a lot of research in developing fast algorithms for frequent pattern mining on text or transactional data [2], [24], trees [25] and graphs [26]. Since the number of candidate patterns to check against a support can at worst be exponential, frequent pattern mining algorithms can be really slow. Consequently distributed versions of these algorithms are of utmost importance. We use the partition-based distributed frequent pattern mining algorithm proposed by Savasere et. al. [27]. The algorithm works by first finding the locally frequent patterns in each partition and then a global scan is required to prune out the false positive patterns. If each partition has a similar number of candidate patterns to evaluate, then depending on the system heterogeneity, faster machines will finish processing faster, however, the overall execution time will be bottlenecked by the slow running partitions. Similar is the case with energy heterogeneity. The partitioning scheme should try to schedule more computation to the machines which have a higher availability of renewable energy. Additionally, a naive partitioning scheme may result in substantial skew in the number of candidate patterns to process in each partition. The execution time for the entire job will increase even if a single partition generates too many patterns. Hence to provide a fully heterogeneity-aware partitioning, the partitions should be homogeneous in terms of the payload characteristics as well. We achieve this by our stratified partitioning strategy which tries to make each partition a representative of the payload.

We evaluate performance in terms of execution time and total dirty energy consumed across all machines. We report results under three different partitioning strategies - 1) Stratified partitioning, 2) Het-Aware ($\alpha = 1.0$) stratified partitioning and 3) Het-Energy-Aware stratified partitioning. A simple random partitioning strategy performs much worse than our baseline (stratified strategy) [28], [3]. For the Het-Energy-Aware scheme, we set the parameter α to 0.999. We will show that by controlling α, we are able to find a solution that simultaneously beats the execution time and the dirty energy footprint of the baseline strategies which create payload partitions of equal sizes. α needs to be set to high values (close to 1.0) to find decent tradeoffs between time and energy as the



(a) Swiss (b) Swiss

(c) Treebank (d) Treebank

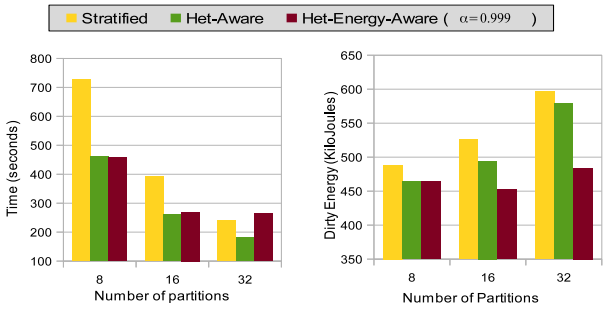
Fig. 2. Frequent Tree Mining on Swiss Protein and Treebank Dataset

two objective functions have different scales. In future, we plan to normalize both objectives to 0-1 range. We ran 2 variants of the frequent pattern mining algorithms:

Frequent Tree Mining: We ran the frequent tree mining algorithm [25] on our 2 tree datasets. Figure 2 reports the execution time and dirty energy consumption on the Swiss protein dataset and the Treebank dataset. Results indicate that using our Het-Aware strategy can improve the runtime by 43% for the 8-partition configuration for Treebank (Figure 2(c)), and this is the best strategy when only execution time is of concern. However, Figures 2(d) and 2(b), that have the energy consumption numbers of these strategies, show that Het-Aware solution is not the most efficient one in terms of dirty energy consumption. Here the Het-Energy-Aware scheme performs the best. In the same 8-partition configuration as described before, the Het-Energy-Aware strategy reduces the execution time by 36% while simultaneously reducing the dirty energy consumption by 9%.

Text Mining: We run the Apriori [2] frequent pattern mining algorithm on the RCV corpus. The execution time numbers are reported in Figure 3(a). Again the Het-Aware scheme is the best with improvement up to 37% reduction in execution time over the stratified partitioning strategy with 8 partitions. The energy numbers are provided in Figure 3(b). The Het-Energy-Aware scheme for the 16-partition configuration reduced the as expected reduced the runtime by 31% while consuming 14% less energy than the stratified partitioning scheme.

2) Graph compression: We test our partitioning scheme on distributed graph compression algorithms. The idea is that we split the input data into p partitions. And then we compress the data in individual partitions independently. We use two



(a) Execution Time (b) Dirty Energy

Fig. 3. Frequent Text Mining on RCV1 corpus

compression algorithms LZ77 [29] and webgraph [30] to compress the data of individual partitions.

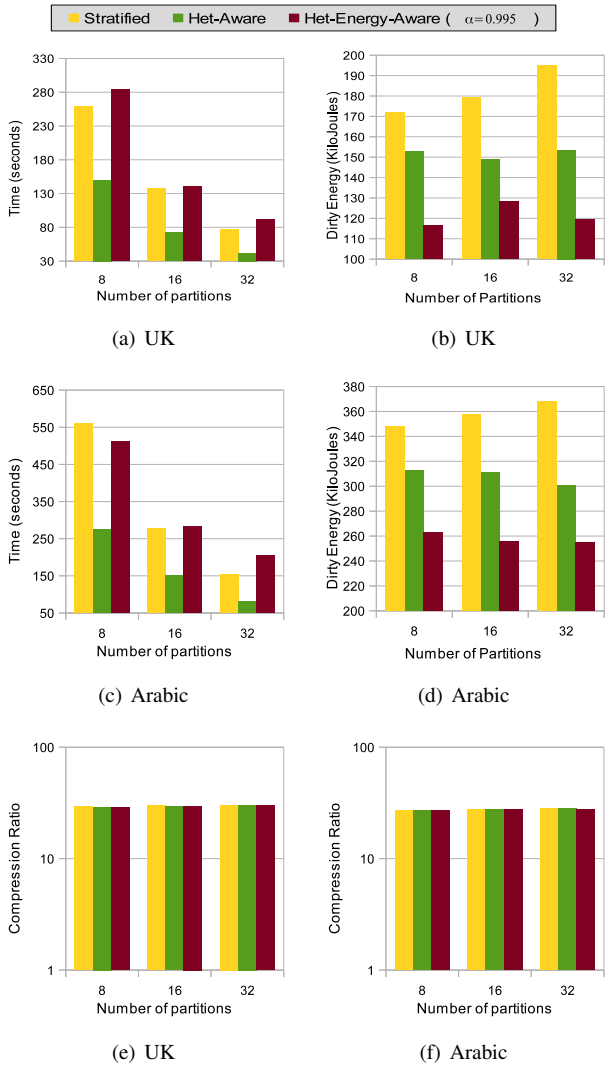
Here we benefit from the partitioning strategy which tries to group similar elements together in a single partition. If a partition comprises of elements which are very similar, then a partition can be represented by a small number of bits. By creating such low entropy partitions, one can get very high compression ratio.

We evaluate performance by the execution time. Again we compare three strategies, stratified partitioning with no heterogeneity awareness, Het-Aware stratified partitioning and Het-Energy-Aware stratified partitioning. Here we set the parameter α to be 0.995 instead of 0.999 as was the case in the frequent pattern mining experiments. Due to reasons explained in Section III-D the execution time should deteriorate quite a bit and should be close to the baselines, while dirty energy consumption rate should improve significantly.

Figure 4 reports the performance numbers (execution time and dirty energy consumption) as well the quality (compression ratio) on both the UK dataset and the Arabic dataset. Our Het-Aware strategy improves the execution time by 51% on the Arabic dataset for the 8-partition configuration (Figure 4(c)). Our Het-Energy-Aware scheme reduces the execution time by only 9%, but simultaneously it reduces the dirty energy consumption by 26% on the same configuration as described above. This also shows the impact of setting a lower α than the frequent pattern mining experiments. The execution time improvements have gone down and dirty energy consumption rate has improved substantially.

We evaluate the quality of our partitioning schemes by comparing the compression ratios achieved by each scheme. Our heterogeneity-aware stratified schemes match the compression ratio of the baseline stratified scheme. Hence we are able to vary the partition sizes to account for better load balancing without any degradation of quality. The technique of reordering the data points according to clusters and creating chunks of variable sizes is able to generate low entropy partitions.

We also run experiments with the very common LZ77 compression algorithm. Tables II and III report the performance and quality numbers for the UK and the Arabic datasets respectively for the 8-partition setting. Lz77 is extremely fast, so there are no gains from our heterogeneity-aware schemes.



(a) UK (b) UK

(c) Arabic (d) Arabic

(e) UK (f) Arabic

Fig. 4. Graph compression results on UK and Arabic webgraphs

Strategy	Time (seconds)	Compression ratio
Stratified	18	18.33
Het-Aware	11	18.2
Het-Energy-Aware	12	18.01

TABLE II. LZ77 COMPRESSION ON UK GRAPH WITH 8 PARTITIONS

The compressibility of our heterogeneity aware techniques is comparable to that of the stratified strategy.

D. Understanding the Pareto-Frontier:

Here we study the effect parameter α has on time-energy tradeoff curve (Pareto-frontier) for all three workloads we consider. For all workloads (Figure 5) we vary the value of α from 1 to 0 and study the impact on execution time and dirty

Strategy	Time (seconds)	Compression ratio
Stratified	38	18.3
Het-Aware	35	18.26
Het-Energy-Aware	40	18.14

TABLE III. LZ77 COMPRESSION ON ARABIC GRAPH WITH 8 PARTITIONS

energy consumption (for 8 partitions). There are two major trends one observes.

First, it is clear that by changing the value of α focus can be effectively shifted from execution time minimization to dirty energy minimization. The magenta line shows this shift. At $\alpha = 1.0$ (extreme left point) execution time is minimum, while dirty energy consumption is maximum for all workloads. This point also represents the Heterogeneity-aware scheme reported earlier. As α is reduced the runtime increases but the dirty energy consumed is reduced. We note that at an α value of about 0.9 dirty energy is typically minimized but at this point the execution time is fairly high. The rationale is that most of the load is placed on the node that harnesses the most green energy leading to severe load imbalance. In other words, at this point the optimizer puts almost all of the payload in the machine with lowest dirty energy footprint. Further lowering α does not have any additional impact.

Second, we observe that the baseline strategy of stratified partitioning is significantly above and to the right of the magenta line (yellow points). Therefore, a simple stratified strategy results in sub-optimal a solution (not Pareto-efficient).

Third, in Figure 6 we evaluated whether our methodology is able to generalize to different parametric settings on the same dataset. We changed the support threshold (a key parameter) for both tree and text datasets and plotted the Pareto frontiers by varying α as described before. For both the datasets, we clearly see that our method is able to find the Pareto frontiers nicely. Hence our framework can tradeoff of between performance and dirty energy across different parametric settings of the same workload. This is particularly important in the context of frequent pattern mining as support is an intrinsic property of the dataset – to find interesting patterns in different datasets, the support has to be adjusted accordingly.

To summarize it is clear that accounting for payload-distribution can result in significant performance and energy gains. Coupled with heterogeneity- and green-aware estimates these gains can be magnified.

VI. Related Works

Data partitioning and placement is a key component in distributed analytics. Capturing of representative samples using the stratified sampling technique on large scale social networks using MapReduce has been investigated by Levin et. al. [31]. Meng [32] developed a general framework for generating stratified samples from extremely large scale data (need not be social network) using MapReduce. Both of these techniques are effective for creating a single representative sample, however, our goal in this work is to partition the data such that each partition is statistically alike. Duong et. al. [28] develops a sharding (partitioning) technique for social networks that performs better than random partitioning. This technique utilizes information specific to social networks to develop effective partitioning strategies. In contrast, our goal is to develop a general framework for data partitioning in the context of distributed analytics. Another related work by Wang et. al. [3] provides a method to mitigate data skew across partitions in the homogeneous context. In this work, we propose to design a framework for heterogeneous context where the heterogeneity is in terms of processing capacity and green energy availability across machines. Performance-aware and energy-aware frameworks are studied extensively in the context of cloud and database workloads [33], [34], [35], [36], [37], [38]. However, these techniques are not payload aware which is extremely critical for large scale analytics workloads. Along with performance and energy skew, data skew also plays a significant role in the performance of analytics tasks.

VII. Concluding Remarks

The key insight we present is that both the quality and performance (execution time and dirty energy footprint) of distributed analytics algorithms can be affected by the underlying distribution of the data (payload). Furthermore, optimizing for either execution time or minimizing dirty energy consumption leads to a Pareto-optimal tradeoff in modern heterogeneous data centers. We propose a heterogeneity-aware partitioning framework that is conscious of the data distribution through a lightweight stratification step. Our partitioning scheme leverages an optimizer to decide what data items to put it which partition so as to preserve the data characteristics of each partition while accounting for the inherent heterogeneity in computation and dirty energy consumption. Our framework also allows data center administrators and developers to consider multiple Pareto-optimal solutions by examining only those strategies that lie on the Pareto-frontier. We run our placement algorithm on three different data mining workloads from domains related to trees, graphs and text and show that the performance can be improved up to 31% while simultaneously reducing the dirty energy footprint by 14% over a highly competitive strawman that also leverages stratification.

Acknowledgments

We thank the anonymous reviewers for their insightful feedbacks. This work is supported in part by NSF grants IIS-1550302, CCF-1629548, CNS-1350941, CSR-1320071 and CNS-1513120.

References

[1] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 720–748, 1999.

[2] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

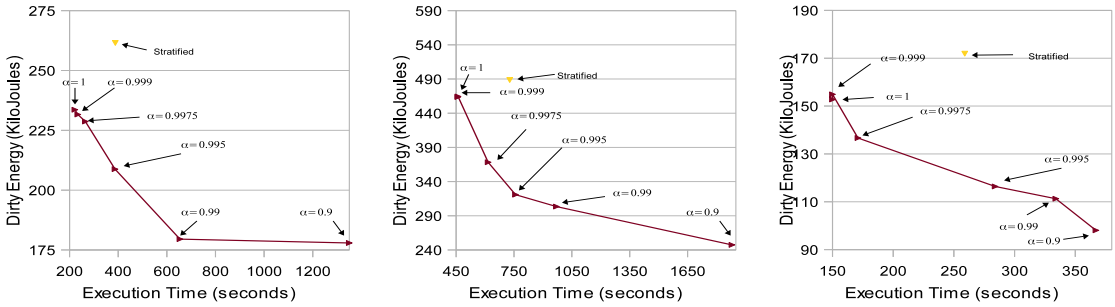
[3] Y. Wang, S. Parthasarathy, and P. Sadayappan, "Stratification driven placement of complex data: A framework for distributed data analytics," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 2013, pp. 709–720.

[4] "Pvwatts simulator," http://pvwatts.nrel.gov/.

[5] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya *et al.*, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Advances in Computers*, vol. 82, no. 2, pp. 47–111, 2011.

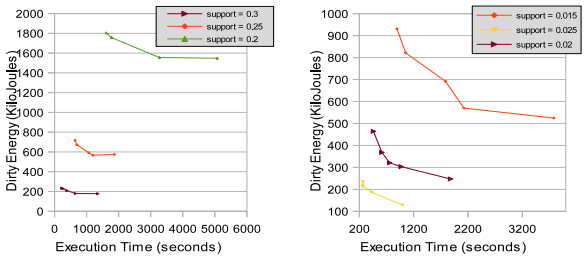
[6] N. Deng, C. Stewart, and J. Li, "Concentrating renewable energy in grid-tied datacenters," in *Sustainable Systems and Technology (ISSST), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 1–6.

[7] C. Li, A. Qouneh, and T. Li, "iswitch: coordinating and optimizing renewable energy powered server clusters," in *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*. IEEE, 2012, pp. 512–523.



(a) Swiss dataset (b) RCV dataset (c) UK dataset

Fig. 5. Pareto frontiers on on a) Tree, b)Text, and c) Graph workloads (8 partitions). Magenta arrowheads represent Pareto-frontier (computed by varying α). Note that both baselines: Stratified (yellow inverted arrowhead); lie above the Pareto frontier (not Pareto-efficient) for all workloads.



(a) Swiss dataset (b) RCV dataset

Fig. 6. Pareto frontiers on a) Tree and b)Text (8 partitions) by changing the support thresholds.

[8] Y. Zhang, Y. Wang, and X. Wang, "Greenware: Greening cloud-scale data centers to maximize the use of renewable energy," in *Middleware 2011*. Springer, 2011, pp. 143–164.

[9] D. Fudenberg and J. Tirole, *Game Theory*. Cambridge, MA: MIT Press, 1991.

[10] J. Choi, D. Bedard, R. J. Fowler, and R. W. Vuduc, "A roofline model of energy," in *27th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2013, Cambridge, MA, USA, May 20-24, 2013*, 2013, pp. 661–672.

[11] S. Parthasarathy, "Efficient progressive sampling for association rules," in *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, 2002, pp. 354–361.

[12] Í. Goiri, R. Beauchea, K. Le, T. D. Nguyen, M. E. Haque, J. Guitart, J. Torres, and R. Bianchini, "Greenslot: scheduling energy consumption in green datacenters," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 20.

[13] H. Prüfer, "Neuer beweis eines satzes über permutationen," *Arch. Math. Phys*, vol. 27, pp. 742–744, 1918.

[14] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 327–336.

[15] T. Bohman, C. Cooper, and A. Frieze, "Min-wise independent linear permutations," *Electronic Journal of Combinatorics*, vol. 7, p. R26, 2000.

[16] A. Chakrabarti and S. Parthasarathy, "Sequential hypothesis tests for adaptive locality sensitive hashing," in *Proceedings of the 24th International Conference on World Wide Web*. ACM, 2015, pp. 162–172.

[17] A. Chakrabarti, B. Bandyopadhyay, and S. Parthasarathy, "Improving locality sensitive hashing based similarity search and estimation for kernels," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer International Publishing, 2016, pp. 641–656.

[18] C.-L. Hwang, A. S. M. Masud, S. R. Paidy, and K. P. Yoon, *Multiple objective decision making, methods and applications: a state-of-the-art survey*. Springer Berlin, 1979, vol. 164.

[19] C. Stewart, A. Chakrabarti, and R. Griffith, "Zoolander: Efficiently meeting very strict, low-latency slos." in *ICAC*, vol. 13, 2013, pp. 265–277.

[20] W. G. Cochran, "Sampling techniques. 1977," *New York: John Wiley and Sons*.

[21] "Uw xml repository," http://www.cs.washington.edu/research/xmldatasets/.

[22] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *The Journal of Machine Learning Research*, vol. 5, pp. 361–397, 2004.

[23] "Law lab datasets," http://law.di.unimi.it/datasets.php/.

[24] M. J. Zaki, S. Parthasarathy, M. Ogihara, W. Li *et al.*, "New algorithms for fast discovery of association rules." in *KDD*, vol. 97, 1997, pp. 283–286.

[25] S. Tatikonda and S. Parthasarathy, "Hashing tree-structured data: Methods and applications," in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*. IEEE, 2010, pp. 429–440.

[26] X. Yan and J. Han, "Closegraph: mining closed frequent graph patterns," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 286–295.

[27] A. Savasere, E. R. Omiecinski, and S. B. Navathe, "An efficient algorithm for mining association rules in large databases," 1995.

[28] Q. Duong, S. Goel, J. Hofman, and S. Vassilvitskii, "Sharding social networks," in *Proceedings of the sixth ACM international conference on Web search and data mining*. ACM, 2013, pp. 223–232.

[29] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *Information Theory, IEEE Transactions on*, vol. 24, no. 5, pp. 530–536, 1978.

[30] P. Boldi and S. Vigna, "The webgraph framework i: compression techniques," in *Proceedings of the 13th international conference on World Wide Web*. ACM, 2004, pp. 595–602.

[31] R. Levin and Y. Kanza, "Stratified-sampling over social networks using mapreduce," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 863–874.

[32] X. Meng, "Scalable simple random sampling and stratified sampling." in *ICML (3)*, 2013, pp. 531–539.

[33] D. Cheng, C. Jiang, and X. Zhou, "Heterogeneity-aware workload placement and migration in distributed sustainable datacenters," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 2014, pp. 307–316.

[34] A. Pavlo, C. Curino, and S. Zdonik, "Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 61–72.

[35] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skewtune: mitigating skew in mapreduce applications," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 25–36.

[36] L. Wang, G. Von Laszewski, J. Dayal, and F. Wang, "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. IEEE, 2010, pp. 368–377.

[37] E. Rahm and R. Marek, "Analysis of dynamic load balancing strategies for parallel shared nothing database systems." in *VLDB*. Citeseer, 1993, pp. 182–193.

[38] Z. Xu, N. Deng, C. Stewart, and X. Wang, "Cadre: Carbon-aware data replication for geo-diverse services," in *Autonomic Computing (ICAC), 2015 IEEE International Conference on*. IEEE, 2015, pp. 177–186.