Traffic At-a-Glance: Time-Bounded Analytics on Large Visual Traffic Data

Xinfeng Li[†*], Gang Li[†*], Fan Yang[†], Jin Teng[†], Dong Xuan[†], Biao Chen[‡]
[†]Dept. of Computer Science & Engineering, The Ohio State University, USA
[‡]Dept. of Computer & Information Science, University of Macau, Macau
{lixinf, lgang, yanfan, tengj, xuan}@cse.ohio-state.edu, bchen@umac.mo

***Abstract*—Massive visual traffic data have become available recently, which provides an opportunity for intelligent traffic analysis. Timely processing is particularly necessary for traffic analysis. In this paper, we study time-bounded aggregation analytics on large visual traffic data. We first find that current MapReduce framework can not work well due to two challenges: first, significant dual diversities exist on data distributions and processing time; second, no apriori knowledge on these distributions and time costs is available. However, we also observe spatial and temporal locality on data values and processing time. Based on the examination, we design TaG, an augmented MapReduce framework for time-bounded traffic analytics jobs. Particularly, we propose a novel sampling algorithm that exploits traffic data localities and stratifies samples based on data distributions and processing time. It runs in an iterative, adaptive manner without apriori knowledge. Moreover, we propose a heuristic scheduling algorithm with considerations of batch processing overhead. Further, we refine load balancing mechanism based on data processing time locality to respect job time bounds. We implement TaG on Hadoop and conduct extensive experiments on a large traffic image dataset. The evaluations on different data sizes show TaG is able to achieve high accuracy within different time bounds.**

I. Introduction

Traffic management is an essential part of smart city [19] and becomes more and more important because of rapid global urbanization. To serve traffic management better, massive traffic cameras are deployed on roads for monitoring [20]. A large set of visual traffic data such as images and videos are produced every day from these cameras. Lots of manpower have been spent to watch and understand such large visual traffic data, and to abstract traffic information, for example, congestion is happening in Broadway street.

Automatic traffic analysis on the large visual traffic data using computer vision techniques can not only save expensive manpower but also enable smarter traffic management. For example, aggregation analytics based on vehicle counts is able to provide historical traffic pattern, detect hot-spots of the city and predict future congestion. Some traffic violation detections can also be performed on the visual traffic data, such as speeding and drunk driving.

For traffic analysis, timely processing is particularly necessary. The longer the processing delay is, the less valuable the analytic results are. For example, less accurate yet more timely vehicle count for the past hour is preferred to predict potential congestion in the coming hour. In addition, to bound the processing time is sometimes necessary for the purpose of saving budget. Because computing resource is not free and visual data processing is especially expensive, it is wise to introduce time bounds. In this paper, we focus on time-bounded aggregation analytics such as SUM, AVG and COUNT on traffic image data.

With time bounds, aggregation analytics become statistical estimations based on processing results of data samples [25], [1]. However, how to approach the best result estimation while respecting time bounds is not trivial. First, dual data diversities on value distributions and processing time costs have to be considered in data sampling and job scheduling. Random sampling is the best strategy for the data that follow one same distribution. But if the data is comprised of multiple diversified distributions, stratified sampling is better. On the other hand, load balancing is highly demanded for the job scheduling with time bounds. Existing big data computing platforms like MapReduce [6] implicitly assume the workloads of Map or Reduce tasks are linear with data sizes. This assumption is usually the case for simple jobs like WordCount, however, the workload per data record becomes highly diversified for traffic analytics jobs. For example, vehicle detection on one image could cost double of the time that is spent on another same size image. So to allocate workloads based on data sizes fails here. Second, lack of apriori knowledge on these distributions and time costs makes the problem even more challenging. Offline prepossessing on visual traffic data is expensive and not desirable.

To address the above challenges, we conduct experiments to study traffic data characteristics and observe spatial and temporal locality on data values and processing time. Specifically, the visual data from the same location and the same time period usually cost similar processing time, and follow similar value distributions. Thus, we assume same distribution for data coming from the same spatio-temporal source, and propose a novel sampling algorithm that refines the estimations of data distributions through iterations and selects data samples adaptively, without apriori knowledge. It determines optimal sample allocation to different variables in an aggregation query

[*]The two authors are co-primary authors. This work is partially supported by the Macau Science and Technology Development Fund under Grant FDCT 092-2014-A2, University of Macau MYRG2015-00165-FST and MYRG112, U.S. National Science Foundation (NSF) grants CNS1065136 and CNS1218876. Any opinions, findings, conclusions, and recommendations in these publications are those of the authors and do not necessarily reflect the views of the funding agencies.

978-1-4673-9953-1/16/$31.00 ©2016 IEEE

by taking into account of diversities in both value distributions and processing time. It is based on the following insight: variables with high variance and low processing time need to be sampled more often to make the overall aggregation deviation small. Further, we design a heuristic job scheduling algorithm that allocates time among sampling iterations, with considerations of batch processing overhead.

In this paper, we embody our sampling and scheduling algorithms in the framework of MapReduce, and design TaG, an augmented MapReduce framework for time-bounded traffic analytics jobs. To serve the aforementioned sampling and scheduling algorithms, we introduce a new online profiling module that collects runtime information and analyzes data distributions. In addition, we refine load balancing mechanism to allocate job workloads based on the data processing time rather than the data size. Therefore, Map or Reduce tasks in TaG may have different number of data records to process but their processing time is expected to be the same. Our load balancing mechanism mitigates "straggler" problem [23] gracefully. Based on our experiments, our load balancing mechanism can speed up job processing by 200%, and guarantee job finish time with only about 5% (1∼3 seconds) deviation.

We implement TaG on Hadoop, a popular open source implementation of MapReduce framework. We examine the life cycle of a MapReduce job in Hadoop and tuned the system thoroughly for much improved time performance so that TaG is able to support jobs with tight time bounds. For practical purposes, we also implement a "caching" mechanism to store processing results which could facilitate subsequent queries on the same data. In this paper, we use vehicle counting in traffic analysis as our case study to demonstrate data diversities and evaluate TaG performance. The results show TaG is able to achieve high accuracy on different data sizes and under different time bounds, for example, 1% error for the SUM query of vehicle counting on 400 GB data within 40 seconds.

In summary, this paper consists of the following key contributions.

- We studied characteristics of visual traffic data processing and summarized challenges and opportunities.
- We proposed a stratified, iterative, adaptive sampling algorithm for aggregation queries given time bounds.
- We designed TaG system based on MapReduce to facilitate time-bounded analytics jobs.
- We implemented TaG on Hadoop and improved system performance in practice.
- We set up real-world clusters and tested the performance of TaG with extensive experiments on a large traffic image dataset.

The rest of this paper is organized as follows. Section II reviews related work. Section III discusses a case study and summarizes challenges and opportunities. Section IV details our system design and Section V presents the system implementation. Section VI reports our experiments and evaluations on TaG. Section VII discusses some open issues with TaG and Section VIII concludes this paper finally.

II. Related Work

Smart city advocates are exploring pervasive Big Data for smart urbanism. For example, large visual data have been processed for citywide surveillance [18]. Traffic analysis is also an indispensable part of smart city, and visual traffic data play an important and unique role in the big data of smart city because of its rich-content feature. Traffic analysis techniques have been studied by many researchers in the past few decades [10], [5] outside the scope of big data. Recently, researchers from computer vision community are leveraging large image dataset like ImageNet [7] for classic vision problems. White *et al.* [22] implement basic computer vision algorithms based on the MapReduce. Compared with TaG, these works do not care about time efficiency and job scheduling.

There is a growing demand to support time-bounded big data aggregation analytics. BlinkDB [1] is a parallel, sampling-based approximate query engine for interactive queries with time bounds. Different from our online sampling, BlinkDB performs offline sampling which relies on historical queries. In addition, the processing time for data with diversified workloads varies significantly, which makes time estimation and meeting deadlines in TaG more challenging. WOHA [16] extends Hadoop to improve deadline satisfaction among MapReduce workflows. Natjam [4] provides hard deadline support in MapReduce with eviction policies. Different from these works, TaG addresses single heavy MapReduce job and handles data sampling.

Stream computing frameworks like Apache S4 [15] and Storm provide timely processing of big data. However, they cannot be directly applied to processing non-streaming data for time-bounded analytics. First, they cannot achieve hard time bounds. When they lack computing resource, either processing delay would be accumulated or some data have to be dropped accidentally. Second, stream computing is based on infinite continuous data stream. Since stream computing is only aware of historic data that have been processed, it can not provide a good result approximation until the end of the data steams. In this paper, we focus on batch processing of the whole dataset rather than stream processing of snippets.

Efficient scheduling algorithms and theoretical analysis are carefully studied in [2], [27], [26]. Grover *et al.* [8] and Laptev *et al.* [13] study iterative MapReduce framework with data sampling. Our TaG is different from them in three places: 1) we seek statistically optimal sampling selection with hard time bounds; 2) our sampling algorithm runs in an iterative manner with runtime profiling and adapts to statistics results in the previous rounds; 3) we focus on visual data and optimize MapReduce for visual data processing characteristics as discussed in Section III.

Data skew Load balancing in MapReduce is also being immensely studied [12], [9], [11], [14]. However, most of them only consider load balancing in the reduce phase. TaG focuses on balancing map task workloads in the very beginning, which is critical for map phase-intensive jobs like traffic analytics.

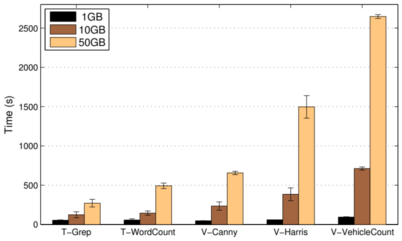

Fig. 1: Computation time of a MapReduce job under different data scale. T: textual data. V: image data.

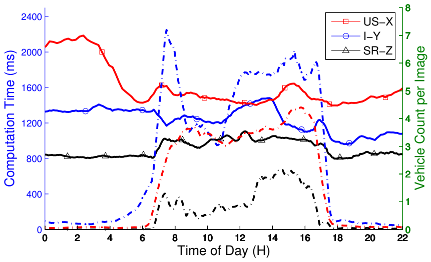

Fig. 2: Computation time (solid line) and result (dash line) of VehicleCount Map task on 1 MB image data.

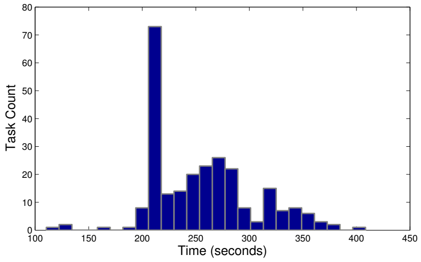

Fig. 3: Distribution of VehicleCount Map task time on 10 GB image data.

Our load balancing is based on data processing time rather than data size. Additionally, TaG is able to learn skewed data distribution without apriori knowledge.

III. Experiencing MapReduce on Visual Traffic Data

A. Case Study: VehicleCount

An exemplary query in the traffic analytics is: "`Get the number of vehicles on Route X on average for the last hour`". The MapReduce paradigm for such a query is shown in Algorithm 1.

Algorithm 1 MapReduce for VehicleCount

```
class Mapper
    method Map(imgSequence $a$, vehicleClassifier $v$)
        for all image $m \in$ imgSequence $a$ do
            $c \leftarrow$ Vehicle-Detection($m$, $v$)
            Emit(cameraID $id$, count $c$)
class Reducer
    method Reduce(cameraID $id$, counts $[c_1,c_2,\cdots]$)
        $sum \leftarrow 0$
        for all count $c \in [c_1,c_2,\cdots]$ do
            $sum \leftarrow sum + c$
        $avg \leftarrow sum$ / SizeOf($[c_1,c_2,\cdots]$)
        Emit(cameraID $id$, count $avg$)
```

B. Characteristics of Visual Traffic Data Processing

First, we measure the overall computation time of a MapReduce job to process visual data on a 21-node cluster. Edge detection, corner detection and Cascade classifier based car detection [17] are performed on traffic images. For comparison, wiki articles as textual data are also tested. The results are shown in Figure 1. Different operations have different time costs, but image data processing is always more time-consuming than textual data processing. For example, HOG based vehicle counting on 50 GB images costs about 2600 seconds, which is more than 5 times the time cost of WordCount on textual data. The time difference is increasing more significantly as the data size becomes larger. Besides, we note that over 90% of time for image-based jobs is spent on the Map phase because of heavy computer vision computations.

Next, we measure the computation time of VehicleCount map tasks on images across one day, whose results are shown in Figure 2. We observe that the processing time of image data processing is highly diversified. For two different cameras' data, their processing time difference can be up to 1300 ms (62%). In addition, we record the histogram distribution of map task time in Figure 3 where the 10 GB image data are are processed on homogeneous machines. Although every map task takes the same size input, their finish time is very different, from 110 seconds to 400 seconds. One explanation on such significant diversity is that computer vision computation varies as visual contexts. Such heterogeneity poses a grave challenge to the load balancing mechanism of MapReduce which implicitly expects an even processing time distribution with evenly-sized splits. MapReduce jobs on textual data do not have such significant processing time diversity. For example, WordCount costs about 300 $\sim$ 380 ms on 1 MB textual data depending on their contents.

Finally, we show the processing results of VehicleCount in Figure 2. We note that the vehicle counts of different cameras during the daytime follow very different distributions with different means and different variations, while the counts during the night are close to 0 consistently.

C. Challenges and Opportunities

We summarize the following challenges of executing time-bounded analytics on visual traffic data.

- Visual data processing is highly time-consuming, and only a small portion of the data can be processed given a tight time bound.
- Processing time of visual data fluctuates significantly. This breaks the naive load balancing mechanism of MapReduce.
- Computation results on different visual data follow different distributions. This makes random sampling algorithm incapable of selecting optimal data samples.
- Prepossessing is not feasible on visual traffic data because of its expensive computation cost and insufficient time for new data. Thus, data distributions on values and processing time are unknown.

However, we also observe spatial and temporal locality on data values and processing time, from our experiments. Specifically, the traffic image data from the same location and the same time period usually cost similar time for processing, and follow similar value distribution. For example, processing time of data from I-Y keeps around 1200 ms with about 10%

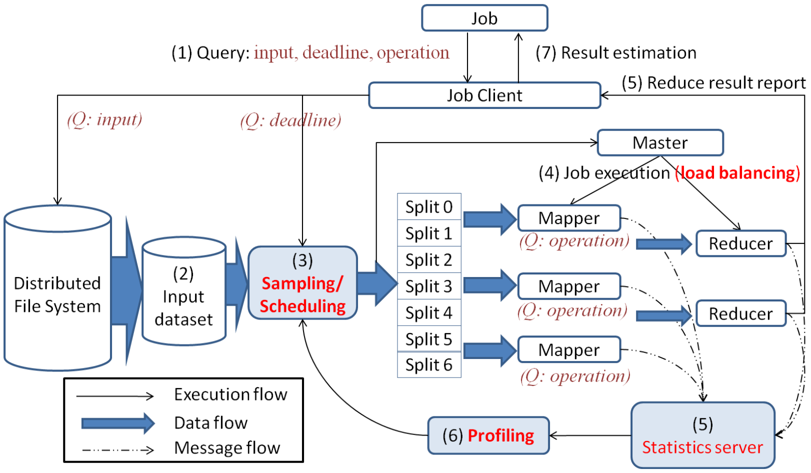

Fig. 4: TaG system architecture and workflow.

difference between daytime and night. Vehicle count daily changes have very similar patterns with two peaks at 7am and 4pm, for example US-X and SR-Z. Thus, we are able to assume same distribution for those data coming from the same spatio-temporal source.

IV. TaG System Design

We design TaG, an augmented MapReduce framework for time-bounded traffic analytics jobs. We detail our system design in this section.

A. System Overview

Figure 4 shows the system architecture and workflow of TaG, where three main modules are carefully designed: sampling, scheduling and load balancing. Next, we give an overview of TaG's execution flow.

1) A TaG job that defines input data (e.g., images from Route No.1), time bound (e.g., 1 minute) and processing operations (e.g., VehicleCount) is submitted.
2) The job client gets the queried dataset from the distributed file system.
3) The scheduling module allocates time budgets to different iterations. And given a time bound, the sampling algorithm selects an optimal subset from the queried dataset based on the profiling information.
4) The master node splits the input data based on an estimation of processing time for load balancing. Then, the job gets executed in mappers and reducers. During job execution, mappers and reducers record input keys, time costs and output results for profiling.
5) After a mapper or a reducer is done, it reports its profiling results to a statistics server on the master node. Reducers also report their aggregated results to the job client.
6) The statistics server aggregates profiling results from mappers and reducers.
7) The job client estimates the final results statistically based on the mean and variance of reducer results, as well as the sample ratio.

Note that both the sampling, scheduling and profiling modules are performed at runtime, so job execution in TaG is an iterative process with more and more refined statistics information.

B. Sampling

As it is often impossible to finish processing all data given time bounds, sampling a data subset is necessary to meet the time deadlines. Sampling is trivial if the data are uniformly distributed, where uniform random sampling can be proved best. However, data uniformity is usually not the case, as we studied in Section III. There are mainly two kinds of data diversities: the computing results of data records contribute differently to the final results, and the time costs of data records are not the same.

First, we formalize the sampling problem with a simple case. Suppose a query is based on the values of n independent variables, for example, $\sum_{i=1}^{n} x_i$, and each variable has a normal distribution, say $x_i \sim N(\mu_i, \sigma_i^2)$. In our traffic analytics case, x_i can be defined as the number of vehicles observed by camera i at one particular time. Due to the time bound, only M observations in total on all the n variables are allowed, assuming making an observation on any of these variables costs the same amount of time. We want to allocate these M observations to different variables, x_i, to get the optimal estimation on $\mu_{\text{sum}} = \sum_{i=1}^{n} \mu_i$. Suppose we make M_i observations on x_i to get $\hat{\mu}_i$ based on M_i observations. Then, $\hat{\mu}_i \sim N(\mu_i, \frac{\sigma_i^2}{M_i})$ and $\hat{\mu}_{\text{sum}} = \sum_{i=1}^{n} \hat{\mu}_i$, so $\text{Var}(\hat{\mu}_{\text{sum}}) = \sum_{i=1}^{n} \text{Var}(\hat{\mu}_i) = \sum_{i=1}^{n} \frac{\sigma_i^2}{M_i}$.

Now, we have the optimization problem as follows.

$$\begin{aligned} \min_{\{M_i\}} \text{Var}(\hat{\mu}_{\text{sum}}) &= \sum_{i=1}^{n} \frac{\sigma_i^2}{M_i} \\ \text{s.t.} \sum_{i=1}^{n} M_i &= M \text{ and } M_i \in \mathbb{N} \end{aligned} \tag{1}$$

If we know all σ_i^2, we have the optimal solution, $M_i = \frac{\sigma_i}{\sum_j \sigma_j} \cdot M$, by the method of Lagrange multipliers. The general rule is to observe high-variance variables more often than low-variance variables.

Next, we consider more practical issues of sampling.

1) Variables x_i are not independent. For example, if x_i and x_j represent two cameras that are only 1 mile away from each other, the value of x_i and x_j are probably correlated.
2) Different M_i have different (time) costs. For example, images from different cameras and/or at different time periods cost different amounts of time.
3) The distribution of variables, σ_i^2, is unknown. This is a more practical and challenging issue for online analytics.

How to deal with variable dependence? Suppose we know σ_i^2, but the variables are not independent. We denote σ_{ij} as the covariance between x_i and x_j. Then, we have

$$\begin{aligned}\mathrm{Var}(\hat{\mu}_{\mathrm{sum}}) &= \sum_{i=1}^{n}\frac{\sigma_i^2}{M_i} + \sum_{i\neq j}\mathrm{Cov}(\sum_{k=1}^{M_i}\frac{x_{ik}}{M_i}, \sum_{k=1}^{M_j}\frac{x_{jk}}{M_j}) \\ &= \sum_{i=1}^{n}\frac{\sigma_i^2}{M_i} + \sum_{i\neq j}M_iM_j\cdot(\frac{1}{M_iM_j})\cdot\mathrm{Cov}(x_i,x_j) \qquad (2) \\ &= \sum_{i=1}^{n}\frac{\sigma_i^2}{M_i} + \sum_{i\neq j}\sigma_{ij}\end{aligned}$$

where x_{ik} and x_{jk} are random variables following the same distributions as x_i and x_j. Notice that the second term is a constant. Thus, the optimal sampling solution to problem (1) remains the same, i.e, $M_i = \frac{\sigma_i}{\sum_j \sigma_j}\cdot M$.

How to deal with different variable costs? We introduce the cost of each variable, t_i, and assume the overall sampling cost budget is fixed, denoted as T. So our optimization problem becomes

$$\begin{aligned}&\min_{\{M_i\}} \mathrm{Var}(\hat{\mu}_{\mathrm{sum}}) = \sum_{i=1}^{n}\frac{\sigma_i^2}{M_i} \\ &\text{s.t. } \sum_{i=1}^{n} t_i\cdot M_i = T \text{ and } M_i \in \mathbb{N}\end{aligned} \qquad (3)$$

With the method of Lagrange multipliers we have the optimal solution:

$$M_i = \frac{\sigma_i}{\sqrt{t_i}\cdot\sum_j(\sigma_j\sqrt{t_j})}\cdot T \qquad (4)$$

How to deal with unknown variances? Basically we want to sample each variable with times in proportion to its standard deviation (we ignore the time cost for now in order to explain our idea clearly). Yet, in reality, we can only get estimates on the deviations, σ_i. If we know that the variables x_1 through x_n conform to some known distributions, we are able to proportionally sample in a statistically correct way. To achieve this, we can use Metropolis-Hastings (MH) sampling [3] to select samples one after another, which finally results in a sampling distribution proportional to the (estimated) variable deviations. In TaG, we propose an iterative, adaptive sampling algorithm based on MH. The essence is to randomly select the next point to sample with a probability proportional to its real probability of occurrence (in this case, its estimated deviation) compared with the current point.

Algorithm 2 SIA Sampling Algorithm

```
Input: Time bound: $T$    Variables: $\{x_k\}$
$t_s \leftarrow$ currentTime
for each variable $x_k$ do
    Sample $M_s$ data records on $x_k$ and process
Update time cost and variance of $\{x_k\}$: $\{t_k\}$, $\{\sigma_k^2\}$
Pick a variable $x_i$ randomly
$t \leftarrow$ currentTime
while $t < (t_s + T)$ do
    Sample $M_p$ data records on $x_i$ and process
    Update time cost and variance of $x_i$: $t_i$, $\sigma_i^2$
    Pick a variable $x_j$ uniformly from $\{x_k\}$
    $\gamma \leftarrow \mathrm{E}(\frac{\hat{\sigma}_j\sqrt{t_i}}{\hat{\sigma}_i\sqrt{t_j}})$
    if $\gamma \geq 1.0$ then
        $x_i \leftarrow x_j$
    else
        $x_i \leftarrow x_j$ with probability of $\gamma$
    $t \leftarrow$ currentTime
```

Suppose we are on one variable x_i now and randomly select the next variable, say x_j. With additional considerations on time costs besides pure deviations, we define $\gamma = \mathrm{E}(\frac{\hat{\sigma}_j\sqrt{t_i}}{\hat{\sigma}_i\sqrt{t_j}})$ where $\hat{\sigma}_i{}^2$ and $\hat{\sigma}_j{}^2$ are estimated variances of x_i and x_j with a certain number of observations. If $\gamma \geq 1$, then we go to x_j to sample; if $\gamma < 1$, then we go to x_j to sample with probability γ. Our proposal jumping distribution is set to a flat one, i.e., uniform distribution among all random variables. According to Markov chain Monte Carlo (MCMC) sampling process, such a tailored MH algorithm will finally approach our objective, i.e., $\frac{M_j}{M_i} \sim \frac{\sigma_j\sqrt{t_i}}{\sigma_i\sqrt{t_j}}$.

Stratified, iterative and adaptive (SIA) sampling algorithm: Assuming no task scheduling overhead, the above sampling process is described in Algorithm 2. Such an iterative, adaptive algorithm allows a dynamic sampling manner where γ changes over time based on gradually refined data variances and time costs. In Algorithm 2, M_s is the number of initial samples to start the sampling process and M_p's value depends on the number of available parallel map slots.

C. Scheduling

SIA Algorithm 2 updates time costs and data variances with frequent iterations assuming little iteration overhead. However, general batch processing platforms like MapReduce have considerable iteration overhead for job initialization and clean-up. The scheduling problem becomes very hard when considering iteration overhead. More iterations benefit better variable distribution estimations, but waste more time in the scheduling overhead; fewer iterations save overhead, but may deviate far from optimal sampling allocation with less precise variable distribution estimations.

In TaG, we propose a heuristic solution as described in Algorithm 3 by approximating SIA Algorithm with fewer iterations, which is good for traffic analytics jobs with tight time bounds. There are three stages. In Stage 1, a few samples are selected to warm up. In Stage 2, we want to limit the deviations of deviations into a small acceptable range. One interesting observation on Equation 4 is the optimal solution is only related to the proportion among σ_i but not their absolute values. By central limit theorem, we know variable $\hat{\sigma}_i$ can be approximated with a normal distribution when the number of observations is large enough. Then, $\hat{\sigma}_i$ has a 95% confidence interval within $\mathrm{E}(\hat{\sigma}_i) \pm 1.96\sqrt{\mathrm{Var}(\hat{\sigma}_i)}$. So if we let $1.96 \cdot \frac{\sqrt{\mathrm{Var}(\hat{\sigma}_i)}}{\mathrm{E}(\hat{\sigma}_i)} \leq \varepsilon$, we are 95% confident that the deviation percentage of $\mathrm{E}(\hat{\sigma}_i)$ is less than ε. In Stage 3, we follow Equation 4 to perform sampling based on latest estimates on time costs and variances. The time allocation between Stage 2 and Stage 3 can be adjusted by η.

Remarks. Our sampling and scheduling algorithms do not assume specific data distributions. For the case of normal distribution, we can get a closed form solution. Please refer to Appendix for the derivation of γ and ξ in the algorithms.

We used SUM (addition) as an exemplary operation for discussion. Yet our sampling algorithm can easily accommodate almost all types of mathematical operations on the statistics derived from the big datasets. For example, AVG, COUNT and subtraction are simple extensions of addition. Multiplications and divisions can be implemented as addition with logarithm of each operand. For a mixed expression with both addition/subtraction and multiplication/division, optimization methods such as conjugate gradient or simulated annealing can be applied to find the optimal or sub-optimal sampling approach on the data.

D. Load Balancing

"Straggler" appears in unbalanced workload splitting. Any mapper straggler prolongs the overall finish time of a job, which is not desired for a time bounded job. As shown in Section III, data processing time may highly depend on their contents. This makes workloads of map task unbalanced if we split input data into evenly-sized chunks as MapReduce. Dynamic load balancing mechanisms [6] that require a much larger number of map tasks do not accommodate time-bounded jobs well because extra system overheads increase as task number increases. Thus, TaG enhanced load balancing mechanism to enable processing time based data splitting.

As shown in Figure 2, the processing time of image data shows temporal and spatial locality. Those images coming from the same camera and within a short time window usually have similar time costs. Thus we are able to estimate the processing time of a sample based on its source and previous statistics information. During the task scheduling, we aggregate the overall estimated processing time for all samples. Once the sampling is done, we first sort the samples based on their sources for data locality. Next, we split the samples into a certain number of splits, each of them having the same estimated processing time. Here, we expect map tasks start and finish working all together at the same time.

To serve load balancing module, as well as sampling and scheduling modules, we introduce profiling to TaG to collect statistics of data processing time and data values. So mappers and reducers in TaG profile such statistics information during executing map and reduce operations. The statistics information is aggregated by a statistics server in the master node. Old profiling results obtained from previous rounds are combined with new ones. Note that TaG profiling module is based on data records which is more fine-grained than existing MapReduce job summary reports, so that we are able to depict detailed distributions on data values and time costs.

Algorithm 3 Three-Stage Scheduling Algorithm

```
Input: Time bound: $T$    Variables: $\{x_k\}$
$t_s \leftarrow$ currentTime
for each variable $x_k$ do
    Sample $M_s$ data records on $x_k$
Process selected samples in MapReduce
Update time cost and variance of $\{x_k\}$: $\{t_k\}$, $\{\sigma_k^2\}$

$t_c \leftarrow 0$
for each variable $x_k$ do
    Let $M_k$ satisfy $\xi = \frac{\sqrt{\mathrm{Var}(\hat{\sigma}_k)}}{\mathrm{E}(\hat{\sigma}_k)} \leq \varepsilon$
    $t_c \leftarrow t_c + M_k \cdot t_k$
$t \leftarrow$ currentTime
if $t_c < \eta \cdot (t_s + T - t) \cdot M_p$ then
    $\kappa \leftarrow 1.0$
else
    $\kappa \leftarrow \frac{\eta \cdot (t_s + T - t) \cdot M_p}{t_c}$
for each variable $x_k$ do
    $M_k \leftarrow \kappa \cdot M_k$
    Sample $M_k$ data records on $x_k$
Process selected samples in MapReduce
Update time cost and variance of $\{x_k\}$: $\{t_k\}$, $\{\sigma_k^2\}$

$t \leftarrow$ currentTime
for each variable $x_k$ do
    $M_k \leftarrow \frac{\sigma_k}{\sqrt{t_k} \cdot \sum_j (\sigma_j \sqrt{t_j})} \cdot (t_s + T - t) \cdot M_p$
    Sample $M_k$ data records on $x_k$
Process selected samples in MapReduce
```

V. Implementation

We have implemented TaG on Hadoop and employ OpenCV for computer vision operations. TaG uses *MapFile* as our input format, in which many data records (e.g., images) are packed into two SequenceFile: one data file and one index file. TaG performs sampling over the index files in memory and seek individual images from a large sequence file easily. The sampling process costs less than 1 second, whose overhead is negligible in the whole MapReduce process. Note that such a data format also helps with the issue of small files (e.g., a large image dataset), a notorious problem in Hadoop. After the keys from the index files are sampled, TaG sorts all samples based on their keys and then splits them into chunks based on estimated processing time. TaG combines separate data records from different MapFiles as a whole split. For the profiling module, we implement a separate server running in the master node to collect statistics information. Mapper and reducer record local statistics and report to the statistics server via a dedicated socket connection.

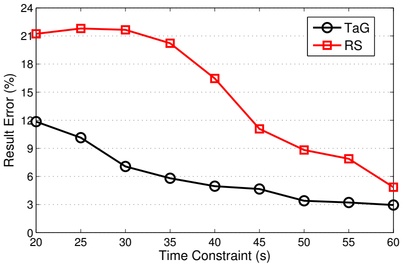

Fig. 5: Accuracy under different time constraints.

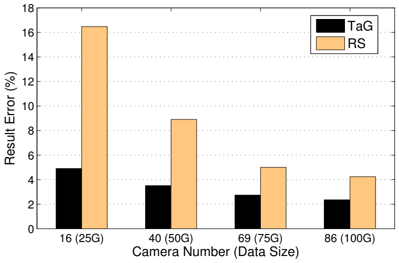

Fig. 6: Accuracy under different data sizes.

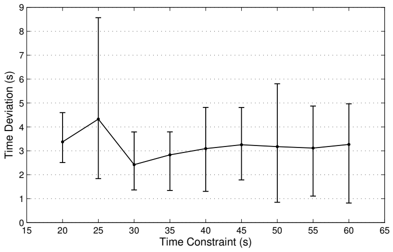

Fig. 7: Finish time deviation under different time constraints.

In addition, we implement a "caching" mechanism in TaG for practical purposes. The intuition of caching in TaG is to cache computing results for complicated jobs like object recognition so the processing time of a subsequent job with the same operations and the same inputs can be saved. We care about the computing results of data records at mappers in TaG, which are stored in a separate directory in HDFS. To utilize cache results, a job called "cache-job" gets executed first before sampling. The cache-job's mappers read cache files and output matched results directly to reducers. The results aggregated by reducers will be finally integrated with normal computing results by the master. Note that cache replacement is not an issue in TaG because the distributed file system is used as storage medium of the cached results.

We have spent great efforts to improve the performance of Hadoop. For example, we disable Job setup and Job cleanup that are not processing data actually. We also enable Java virtual machine (JVM) reuse which can reduce JVM launch overhead for every new task. Furthermore, we shorten the heartbeat interval to speed up task scheduling, which is particularly beneficial for jobs with tight time bounds.

VI. Evaluation

We evaluate the TaG system on a large traffic image dataset in real-world clusters. We report the evaluation results in this section.

A. Experiment Setup

We set up two kinds of clusters: a homogeneous one and a heterogeneous one. The homogeneous cluster consists of 11 machines where 1 workstation (4 cores, 16 GB RAM) serves as the master node and 10 Dell machines (2 cores, 4 GB RAM) serve as worker nodes. The heterogeneous cluster has 21 heterogeneous machines, consisting of 1 workstation, 10 Dell machines, 5 Lenovo machines (4 cores, 4 GB RAM) and 5 Powerspec machines (2 cores, 2 GB RAM).

For our traffic analytics case, we collected around 560 GB images from online traffic cameras. To make this traffic dataset comprehensive, we collected the data across different time periods of a day and across both weekdays and weekends. Together, the test dataset consists of 396 cameras and 15,992,845 images. In our experiment, the test case is VehicleCount as discussed in Section III. Note that vehicle detection result is deterministic and we treat the detection result as the ground truth. The accuracy of vehicle detection algorithm is not the focus of this paper.

B. Evaluation Results

Next, we show the evaluation results of TaG on a variety of metrics, including accuracy of computing results, finish time deviation against time bound, improvement of load balancing and caching, as well as the performance under heterogeneous environments. For comparison, we also tested a random mechanism which runs iteratively with random time budgets and picks stratified samples across variables in each round (denoted as RS mechanism). We repeated each experiment 50 times.

1) Accuracy: Accuracy is defined as result error, the deviation percentage of estimated results against the ground truth. First, we measured the accuracy of different mechanisms under different time bounds. This experiment is performed on a 25 GB size of daytime images from 16 cameras. The results are shown in Figure 5. We can see that TaG outperforms the RS mechanism by up to 3 times. The accuracy of TaG increases fast as the time bound increases. Finally, the accuracy of the RS mechanism approaches to that of TaG when the time bound is over 60 seconds. This is because more samples make both results closer to the ground truth. But TaG is still better than the RS mechanism in terms of their relative accuracy.

Second, we measured the accuracy of different mechanisms under different data sizes. The time bound is set to 40 seconds. In Figure 6, we can see that the accuracy of both TaG and the RS mechanism decreases as the number of cameras (i.e., data size) increases. This is mainly because the data from some cameras with low variances get involved and the overall variance against the mean decreases relatively. This shows our TaG favors data with high and diversified variances. TaG is better than the RS mechanism, with 1/2 $\sim$ 2/3 less errors.

Data Size	25 GB	50 GB	75 GB	100 GB
Average (s)	3.09	2.31	0.91	3.65
90 percentile (s)	4.81	3.78	1.84	9.56
10 percentile (s)	1.31	0.76	0.22	0.91

TABLE I: Finish time deviation under different data sizes.

2) Finish Time Deviation: We show the finish time deviation of TaG in Figure 7 and Table I. In general, TaG is able to finish a job with a roughly 3 second deviation, which is satisfactory. The causes of the deviation include inconsistent processing time of different cameras and uncertain system

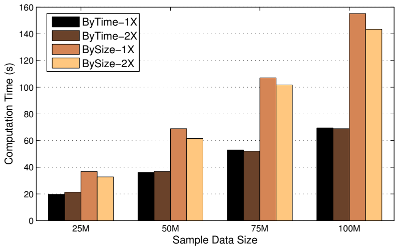

Fig. 8: Job computation time of different load balancing policies.

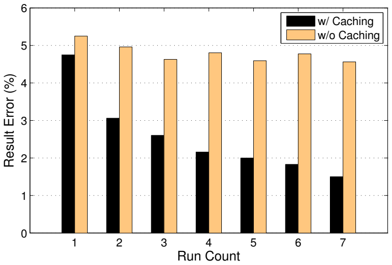

Fig. 9: Accuracy improvement with caching.

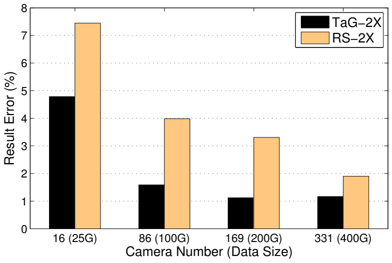

Fig. 10: Accuracy on a heterogeneous cluster.

extra overhead like scheduling delay. Note that we set the number of map tasks as 1X of the available map slots. So this performance is achieved by our processing time estimation and load balancing mechanism, rather than by many iterations with short jobs. As shown in Figure 7, the finish time deviation increases slowly as the time bound increases. The peak at 25 seconds is because the first warm-up round costs about 15 seconds, and our algorithm may not work properly with the rest time considering an extra system cost of 7 $\sim$ 10 seconds.

Table I shows the finish time deviation of TaG under different data sizes (number of cameras) with a fixed time bound of 40 seconds. As camera number increases, the first round takes more time and the time budget left for the 2nd and 3rd rounds decreases, so the error for TaG's time estimation also decreases. But the time budget becomes less than extra system cost when the data size is 100 GB and make TaG algorithm malfunction, just as the above-mentioned case.

3) Load Balancing: In Figure 8, we measured the job computation time of different load balancing policies. Here the computation time is defined as the time period from the first mapper start to the last mapper completion. Reduce time is not considered here since it is very little. We compared load balancing based on the processing time or data size. For each policy, we set the number of map tasks to either 1X or 2X of the total map slots on the homogeneous cluster. We can clearly see the improvement of the time-based load balancing policy against traditional size-based load balancing in Figure 8. Our load balancing policy can shorten the overall computation time by 1/3 to 1/2. In other words, a job in TaG is able to be speeded up by 2 times.

Figure 9 shows the improvement of TaG on accuracy with our caching module. We repeated the same query on 25 GB data with a 40 second time bound multiple times and measured the accuracy change with or without caching. Caching is able to increase the accuracy gradually as more and more samples in the previous rounds are cached and stored in HDFS. We notice that the change from the 1st run and the 2nd run is significant. This is because the estimation in the 2nd run is actually based on the almost doubled number of accumulated samples from the the 1st run. Without caching, the error rate fluctuates around a high value.

Although the current TaG is mainly designed on homogeneous clusters, we tested its performance in the 21-node heterogeneous cluster. We set the number of map tasks as 2 times of the total map slots. With more short map tasks, TaG is able to mitigate the influence of work node heterogeneity to a large extent. The RS mechanism with size-based load balancing is compared, whose map task number is also set to 2X. The time bound is set to 40 seconds. Figure 10 shows the results. Compared with the result in Figure 6, TaG performs a bit worse on a heterogeneous cluster. However, TaG still works better than the RS mechanism. This means although the estimation of data processing time becomes imperfect, TaG still gives higher sampling weights on high variance data to have a better overall result estimation.

It is worth noting that TaG can fit a heterogeneous environment well because our profiling module is able to measure heterogeneity of machine computing capabilities on the fly. By incorporating approaches like ActCap [21], TaG would have a good estimation on data processing time in different machines. In this way, our load balancing module becomes aware of data diversity and node heterogeneity in data placement.

VII. Discussion

There are many future extensions we can make based on our TaG design.

- **Extension to general data processing.** Although we focus on traffic analysis in the paper, diversified processing time and data values are a general problem in processing visual data and some other non-trivial data, for example face detection and counting on social networking photos, and sequencing analysis in bioinformatics. There will be no explicit spatio-temporal locality to utilize for general uneven data thus extra efforts to cluster data distributions have to pay.
- **Extension to other big data computing platforms.** Current TaG is implemented on Hadoop, a MapReduce framework. However, our design modules in TaG are also applicable to other platforms. Spark [24] leverages memory to store intermediate results, which facilitates iterative MapReduce natively. With little iteration overhead, our SIA sampling Algorithm 2 could be applied, with some modification to the profiling module so that statistics on data distributions can be updated in real time. Figure 11 compares accuracy performance of different sampling algorithms with simulation. We can see that our SIA algorithm approaches to the optimal one, which knows distributions and stratifies on both values and time, very quickly as total cost budget increases, and becomes much better than the random algorithm.

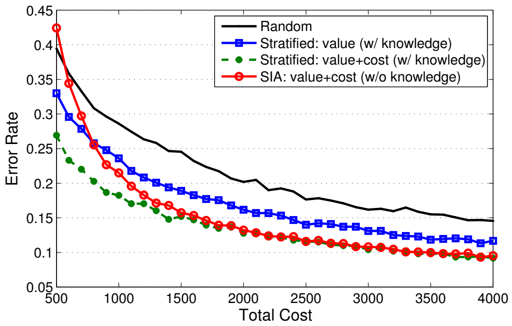

Fig. 11: Accuracy of different sampling algorithms for SUM query on two variables: $X_1 \sim N(5, 10^2)$, $cost_1 = 5$ and $X_2 \sim N(5, 5^2)$, $cost_2 = 100$.

VIII. Conclusion

In this paper, we present our TaG system, an augmented MapReduce framework for time-bounded traffic analytics jobs. In TaG, we propose a stratified sampling algorithm that considers data diversities on both values and time costs and runs in an iterative, adaptive manner without apriori knowledge on data distributions and a heuristic scheduling algorithm with practical consideration of iteration overhead. We also propose a processing time based load balancing mechanism to speed up job execution. We implement TaG on Hadoop and test it on a large traffic image dataset. The evaluation results demonstrate the efficiency and robustness of TaG.

Acknowledgment

We would like to thank Prof. Yuan F. Zheng, Adam C. Champion, Sihao Ding, and Qiang Zhai for their assistance with the experiments and insightful discussions.

References

[1] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *EuroSys*, pages 29–42, 2013.

[2] H. Chang, M. Kodialam, R. Kompella, T. Lakshman, M. Lee, and S. Mukherjee. Scheduling in mapreduce-like systems for fast completion time. In *INFOCOM*, pages 3074–3082, 2011.

[3] S. Chib and E. Greenberg. Understanding the metropolis-hastings algorithm. *The American Statistician*, 49(4):327–335, 1995.

[4] B. Cho, M. Rahman, T. Chajed, I. Gupta, C. Abad, N. Roberts, and P. Lin. Natjam: Design and evaluation of eviction policies for supporting priorities and deadlines in mapreduce clusters. In *SoCC*. ACM, 2013.

[5] B. Coifman, D. Beymer, P. McLauchlan, and J. Malik. A real-time computer vision system for vehicle tracking and traffic surveillance. *Transportation Research Part C: Emerging Technologies*, 6(4):271–288, 1998.

[6] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[8] R. Grover and M. J. Carey. Extending map-reduce for efficient predicate-based sampling. In *ICDE*, pages 486–497, 2012.

[9] B. Gufler, N. Augsten, A. Reiser, and A. Kemper. Load balancing in mapreduce based on scalable cardinality estimates. In *ICDE*, pages 522–533, 2012.

[10] J.-W. Hsieh, S.-H. Yu, Y.-S. Chen, and W.-F. Hu. Automatic traffic surveillance system for vehicle tracking and classification. *Intelligent Transportation Systems, IEEE Transactions on*, 7(2):175–187, 2006.

[11] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi. Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud. In *CloudCom*, pages 17–24, 2010.

[12] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. Skew-resistant parallel processing of feature-extracting scientific user-defined functions. In *SoCC*, pages 75–86. ACM, 2010.

[13] N. Laptev, K. Zeng, and C. Zaniolo. Early accurate results for advanced analytics on mapreduce. *PVLDB*, 5(10):1028–1039, 2012.

[14] Y. Le, J. Liu, F. Ergün, and D. Wang. Online load balancing for mapreduce with skewed data input. In *INFOCOM*, 2014.

[15] N. Leonardo, R. Bruce, N. Anish, and K. Anand. S4: Distributed stream computing platform. In *ICDMW*, pages 170–177, 2010.

[16] S. Li, S. Hu, S. Wang, L. Su, T. Abdelzaher, I. Gupta, and R. Pace. Woha: Deadline-aware map-reduce workflow scheduling framework over hadoop clusters. In *Proceedings of IEEE ICDCS*, 2014.

[17] D. Ponsa and A. López. Cascade of classifiers for vehicle detection. In *Advanced Concepts for Intelligent Vision Systems*, pages 980–989. Springer, 2007.

[18] J. Teng, J. Zhu, B. Zhang, D. Xuan, and Y. F. Zheng. Ev: efficient visual surveillance with electronic footprints. In *INFOCOM*, 2012.

[19] IBM Smart City. http://www.ibm.com/smarterplanet/us/en/traffic_congestion/ideas/. [Online].

[20] TrafficLand. http://www.trafficland.com/. [Online].

[21] B. Wang, J. Jiang, and G. Yang. Actcap: Accelerating mapreduce on heterogeneous clusters with capability-aware data placement. In *INFOCOM*, 2015.

[22] B. White, T. Yeh, J. Lin, and L. Davis. Web-scale computer vision using mapreduce for multimedia data mining. In *MDMKDD*, page 9, 2010.

[23] H. Xu and W. C. Lau. Optimization for speculative execution in a mapreduce-like cluster. In *INFOCOM*, 2015.

[24] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *HotCloud*, 2010.

[25] K. Zeng, S. Gao, J. Gu, B. Mozafari, and C. Zaniolo. Abs: a system for scalable approximate queries with accuracy guarantees. In *SIGMOD*, pages 1067–1070. ACM, 2014.

[26] Y. Zheng, N. B. Shroff, and P. Sinha. A new analytical technique for designing provably efficient mapreduce schedulers. In *INFOCOM*, 2013.

[27] Y. Zhu, Y. Jiang, W. Wu, L. Ding, A. Teredesai, D. Li, and W. Lee. Minimizing makespan and total completion time in mapreduce-like systems. In *INFOCOM*, 2014.

Appendix: γ and ξ in Sampling Algorithms

Suppose we have already made K observations on variable x that follows $x \sim N(\mu, \sigma_i)$. We can get the estimated mean $\hat{\mu} = \frac{\sum_i X_i}{K}$ and variance $\hat{\sigma^2} = \frac{\sum_i (X_i - \hat{\mu})^2}{K-1}$. Now, $\hat{\mu}$ is still a normal random variable, and $\hat{\sigma^2}$ complies with a Gamma distribution.

$$\mathrm{E}(\hat{\sigma^2}) = \frac{\alpha}{\beta} = \sigma^2, \ \mathrm{Var}(\hat{\sigma^2}) = \frac{\alpha}{\beta^2} = \frac{2\sigma^4}{K-1} \tag{5}$$

where $\alpha = \frac{K-1}{2}$ and $\beta = \frac{K-1}{2\sigma^2}$. With some computations, we have

$$\begin{gathered}\mathrm{E}(\hat{\sigma}) = \frac{1}{\sqrt{\beta}} \cdot \frac{\Gamma(\alpha + \frac{1}{2})}{\Gamma(\alpha)}, \ \mathrm{E}(\frac{1}{\hat{\sigma}}) = \sqrt{\beta} \cdot \frac{\Gamma(\alpha - \frac{1}{2})}{\Gamma(\alpha)} \\ \mathrm{Var}(\hat{\sigma}) = \frac{\alpha}{\beta} - \mathrm{E}(\hat{\sigma})^2\end{gathered} \tag{6}$$

Thus,

$$\xi = \frac{\sqrt{\mathrm{Var}(\hat{\sigma}_i)}}{\mathrm{E}(\hat{\sigma}_i)} = \frac{\Gamma(\alpha)}{\Gamma(\alpha + \frac{1}{2})} \cdot \sqrt{\alpha - \frac{\Gamma(\alpha + \frac{1}{2})}{\Gamma(\alpha)}^2} \tag{7}$$

and

$$\gamma = \mathrm{E}(\frac{\hat{\sigma}_j \sqrt{t_i}}{\hat{\sigma}_i \sqrt{t_j}}) = \sqrt{\frac{t_i \beta_i}{t_j \beta_j}} \cdot \frac{\Gamma(\alpha_j + \frac{1}{2})}{\Gamma(\alpha_j)} \cdot \frac{\Gamma(\alpha_i - \frac{1}{2})}{\Gamma(\alpha_i)} \tag{8}$$