A Framework for Real-Time Information Derivation from Big Sensor Data

Liehuo Chen and Kyoung-Don Kang
Department of Computer Science
State University of New York at Binghamton
{lchen66,kang}@binghamton.edu

***Abstract*—In data-intensive real-time applications, e.g., transportation management and location-based services, the amount of sensor data is exploding. In these applications, it is desirable to extract value-added information, e.g., fast driving routes, from sensor data streams in real-time rather than overloading users with massive raw data. However, achieving the objective is challenging due to the data volume and complex data analysis tasks with stringent timing constraints. Most existing big data management systems, e.g., Hadoop, are not directly applicable to real-time sensor data analytics, since they are timing agnostic and focus on batch processing of previously stored data that are potentially outdated and subject to I/O overheads. To address the problem, we design a new real-time big data management framework, which supports a non-preemptive periodic task model for continuous in-memory sensor data analysis and a schedulability test based on the EDF (Earliest Deadline First) algorithm to derive information from current sensor data in real-time by extending the map-reduce model originated in functional programming. As a proof-of-concept case study, a prototype system is implemented. In the performance evaluation, it is empirically shown that all deadlines can be met for the tested sensor data analysis benchmarks.**

I. Introduction

In a number of important real-time applications, e.g., transportation management, location-based services, and structural health monitoring, the volume of sensor data is increasing rapidly. It is required to process large amounts of sensor data in real-time to extract value-added information, e.g., fast/fuel-efficient driving routes and user mobility/network usage patterns. However, timely extraction of valuable information from large raw sensor data is challenging due to the increasing data size and complex data analysis tasks with timing constraints.

MapReduce [1] and Hadoop [2] greatly simplify the development of parallel big data analysis applications.[1] A user only has to write serial `map()` and `reduce()` functions. The underlying runtime system divides massive data into smaller chunks and schedules map/reduce tasks to process the data chunks in parallel on the user's behalf. However, they are not readily applicable to real-time sensor data analytics for several reasons. First, they are timing agnostic. As a result, they may miss many deadlines, diminishing the value of the derived information. They only support one-time batch processing of the data at rest stored in the distributed file system. Thus, the data could be outdated and subject to significant I/O overheads. Any information, e.g., route recommendations, derived using stale sensor data has little value. Moreover, not the system but a user has to manually execute sensor data analytics periodically, if necessary, to continuously analyze the real world status. This approach is tedious and further increases the difficulty of developing real-time sensor data analysis applications.

Despite the increasing demand for real-time sensor data analytics, related work is relatively scarce. Advanced data stream management systems, e.g., Storm [4], S4 [5], and Spark Streaming [6], support *near* real-time stream data processing; however, they do not consider explicit deadlines or real-time scheduling to ensure the timeliness of data processing. Although the problem of meeting deadlines in Hadoop has been investigated [7], [8], [9], [10], they inherit the short-comings of Hadoop optimized for batch processing of the data in the secondary storage. To address the problem, we design a new real-time map-reduce framework, called RTMR (Real-Time Map-Reduce), that provide several unique features not provided by most existing big data management systems including [1], [2], [4], [5], [6], [11], [7], [8], [9], [10]:

- Using the API (Application Programming Interface) of RTMR, a user − an application developer − can write serial `map()` and `reduce()` functions for a specific real-time data analysis application, and specify the data analysis task parameters, e.g., the deadlines and periods.
- A non-preemptive periodic task model is supported for continuous real-time analysis of sensor data. Moreover, an EDF-based schedulability test is provided to support timing constraints considering both the computation and data access delay.
- Several mechanisms for efficient in-memory sensor data analysis are supported. First, sensor data are directly streamed into main memory to let RTMR distill information from them on the fly. Second, intermediate data generated in a map/reduce phase is pipelined straight to the next phase, if any, without being staged in the local disk or distributed file system unlike Hadoop and its variants. Further, memory reservation is supported to ensure enough space is allocated to store the input, intermediate, and output data for each real-time sensor

[1]Even though there is no single definition of big data on which everybody agrees, the notion of five Vs of big data [3] − volume, velocity, variety, veracity (uncertainty), and value − is broadly accepted; that is, the volume, variety, and velocity of data generation are increasing fast. Also, from big data that may involve uncertainties, valuable information needs to be extracted.

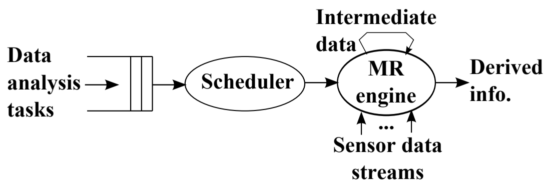

Fig. 1. RTMR Structure

data analysis task.

In addition, we have implemented RTMR by extending Phoenix++ [11], a state-of-the-art open source multicore map-reduce framework, to support the aforementioned RTMR features for real-time data analytics rather than retrofitting Hadoop.

For performance evaluation, we generate synthetic periodic workloads using four micro-benchmarks to model real-time data analysis tasks: k-means clustering, linear regression, histogram, and matrix multiplication that can be applied to support, for example, mobile user-clustering for location-based services, sensor data value or financial market prediction, and autonomous vehicles. Using the benchmarks, we design several real-time data analysis task sets and analyze their schedulability. For the task set with the tightest deadlines, the performance evaluation results empirically verify the schedulability test by showing that all deadlines are actually met. On the other hand, Phoenix++ used as the baseline fails to do it. Further, for the task set, RTMR processes over 0.72 TB of sensor data in each 1000 second experimental run, which translates to more than 2.59 TB/hour and 62 TB/day.

The remainder of this paper is organized as follows. In Section II, an overview of the RTMR architecture is given. In Section III, the real-time task model and scheduling for real-time data analytics are described. In Section IV, the performance of our approach is evaluated. Related work is discussed in Section V. Finally, Section VI concludes the paper and discusses future work.

II. Overall Architecture of RTMR

The overall structure of our framework for real-time data analytics is depicted in Figure 1. The periodic instances of the data analysis tasks, called jobs, are scheduled via the EDF scheduler. The job dispatched by the scheduler is processed by the map-reduce (MR) engine until the completion without being preempted to avoid large overheads for preemption and context switching in real-time data analytics. Although it is often assumed that the context switch overhead is ignorable in real-time scheduling, this may not be the case for real-time data analysis tasks, each of which deals with relatively large data every period, e.g., millions of sensor readings per period for transportation management or location-based services. As illustrated in Figure 1, sensor data are directly streamed into main memory and intermediate data are pipelined to the next map/reduce phase, if any, until the job is completed. Finally, the derived information is returned to the user.

```
// input: (CellID, PhoneNum) pairs
// output: intermediate (key, value) pairs
map(void *input) {
  for each CellID in input {
      emitIntermediate(CellID,1);
  }
}
```

Fig. 2. Map Function for Cell Phone Count

```
// input: intermediate (CellID, 1) pairs
// output: (CellID, count)
reduce(int key, iterator value) {
  int count = 0;
  for each v in value {
      count = count + v;
  }
  emit(key, count);
}
```

Fig. 3. Reduce Function for Cell Phone Count

In RTMR, a user has to write two serial functions, map() and reduce(), and specify the period and deadline for each real-time data analysis task. For example, the map and reduce functions in Figures 2 and 3 can be used to periodically monitor the number of the active mobile phones in each cell in a local cellular network as a basis to analyze customers' mobility and network usage patterns.

Figure 4 illustrates the processing steps and data flow of the MR engine in Figure 1 that processes real-time sensor data analysis tasks based on the map-reduce model, which is originated in functional programming [12] but not tied to a specific implementation, e.g., Hadoop. More specifically, the earliest deadline job dispatched by the scheduler is processed by the MR engine as follows.

1) In RTMR, each input sensor datum is expressed as a (key, value) pair, e.g., (cell ID, phone number) for location-based services, and streamed into memory. The input (key, value) pairs are evenly divided into chunks

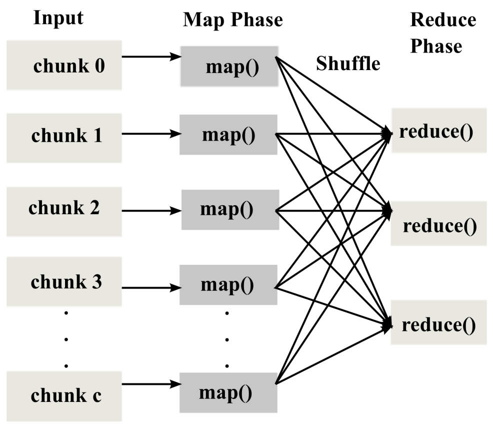

Fig. 4. Map-Reduce Model

by the MR engine and assigned to mappers, i.e., worker threads.

2) The mappers independently execute the user-specified map() function on different data chunks in parallel. For example, each mapper executes the map() function for cellphone count in Figure 2, producing intermediate (key, value) pairs as a result.
3) The map phase is completed when all the mappers finish processing the assigned data chunks. If there is no reduce phase, which is optional, the (key, value) pairs produced by the mappers are returned as the final result and the job is terminated.
4) If there is a reduce phase, the intermediate (key, value) pairs produced by the mappers are directly pipelined to one or more reducers and sorted based on their keys. Specifically, the pointers to the intermediate results in memory are passed to the reducers with no expensive data copies.
5) The reducers execute the user-defined reduce() function in parallel to produce the final (key, value) pairs by processing the assigned non-overlapping intermediate (key, value) pairs. When all the reducers complete, the final (key, value) pairs are returned and the job is terminated. In an iterative application that consists of multiple pairs of map and reduce phases, the output of the reduce phase is directly pipelined to the map phase of the next iteration by passing the pointers to the data.

In RTMR, all input, intermediate, or final (key, value) pairs are stored in memory unlike MapReduce [1], Hadoop [2], or their variants. Phoenix++ [11] effectively utilizes the memory hierarchy to process map-reduce tasks in memory using multiple CPU cores. However, it does not support a periodic task model, real-time scheduling, direct streaming of sensor data into memory, or memory reservation. Instead, it only supports FIFO scheduling. Further, it reads input data from and writes output to the disk. RTMR extends Phoenix++ by supporting: 1) input sensor data streaming, 2) intermediate data pipelining, 3) a non-preemptive periodic task model, 4) memory reservation, and 5) an EDF-based schedulability test and scheduling required for real-time data analytics. A detailed description of the real-time task model, memory reservation, and scheduling follows.

III. Real-Time Task Model and Scheduling

In this section, the task model, memory reservation, and scheduling supported by RTMR are discussed.

A. Task Model and Memory Reservation

In this paper, we assume that a real-time sensor data analysis system needs to execute a set of n independent periodic map-reduce tasks $\Gamma = (\tau_1, \tau_2, ..., \tau_n)$ that are not self-suspending. In the system, there are $m \geq 1$ cores available for real-time data analytics. In this paper, an arbitrary real-time map-reduce task $\tau_i \in \Gamma$ is associated with the period P_i and relative deadline $D_i = P_i$ (implicit deadline). τ_i is a real-time data analysis task that consists of s_i (≥ 1) parallel execution segments, i.e., map/reduce phases, defined as follows:

Definition 1:

$$\tau_i : ((< e_i^1, m_i^1 >, ..., < e_i^{s_i}, m_i^{s_i} >), C_i, D_i)$$

where s_i = 1, if τ_i only consists of a map phase. s_i = 2, if it has both map and reduce phases. $s_i > 2$ if it consists of multiple pairs of parallel map and reduce phases iteratively executed in sequence. In addition, e_i^j and m_i^j are the estimated maximum execution time of segment j and the number of cores used in the segment, respectively. The (estimated) maximum execution time of τ_i is: $C_i = \sum_{j=1}^{s_i} e_i^j$.

Using the API of RTMR, a user needs to specify the map() and reduce() functions as well as s_i and D_i in Definition 1 for τ_i considering the application semantics.[2] In RTMR, C_i is estimated offline considering not only the CPU time but also the memory access delay, because the data access delay may not be ignorable in real-time data analytics. In this paper, $\tau_i \in \Gamma$ is run multiple times offline. For each run of τ_i, the latency from reading the first input (key, value) pair to producing the last output (key, value) pair is used as the estimated execution time to consider both the computation and data access latency. The maximum observed execution time acquired from the prespecified number of runs is used as C_i.[3]

In RTMR, input sensor data are streamed into memory as discussed before. Further, we assume that the size of input sensor data is predetermined. At the end of a segment, τ_i produces intermediate data that is input for the next segment executed consecutively. If there is no following segment, they are the final output of a periodic instance of τ_i. Given that, RTMR analyzes the maximum intermediate/output data sizes and reserves enough memory for τ_i, which typically consists of a few common operations, such as filtering, aggregation, or prediction of physical phenomena (e.g., the traffic speed in a road segment) based on the recent history. If unimportant input data are filtered out or sensor data are aggregated in a phase, the size of the output/intermediate data produced at the end of the phase is not bigger than the input. Prediction via, for example, linear/nonlinear regression produces a predefined number of model parameters, which is considerably smaller than the input. Even when a join, one of the most expensive operator for data processing, is performed between a pair of input sensor data of sizes N and M, the maximum output size is limited to NM in the worst-case. Also, N and M are relatively small compared to data sizes considered in batch data analysis systems, e.g., Hadoop, because only the current sensor data are processed per period for real-time data analytics in RTMR.

[2]For the clarity of presentation, we assume that the data partitioning and shuffle steps in Figure 4 are included in the map and reduce phases, respectively. Also, their latencies are added to the execution times of the map and reduce phases.

[3]In general, the analysis of worst-case execution times in a multicore processor is an open problem. Analyzing the execution times of real-time data analysis tasks is even more challenging due to the increasing size and volatility of sensor data. A thorough investigation of more advanced approaches to analyze the execution times of real-time data analysis tasks is reserved for future work.

B. Schedulability Test

In the j^{th} segment of τ_i, where $1 \leq j \leq s_i$, m_j threads are used to run the user-specified map() or reduce() function of τ_i in a parallel segment depending on whether τ_i is currently in the map or reduce phase. In this paper, each core runs a single map or reduce thread at a time. However, m_j threads run in parallel in the j^{th} segment, following the data-parallel, single-instruction-multiple-data (SIMD) model. In parallel real-time data analytics, there is a trade-off between data and task parallelism. If more cores are used by an individual task to process more data simultaneously in a SIMD manner, fewer tasks can run in parallel or vice versa. As scheduling in multiprocessor real-time systems is NP-hard in the strong sense [13], we devise a heuristic to schedule real-time data analysis tasks in this paper. More specifically, we intend to maximize the data parallelism subject to the available hardware parallelism by setting $m_i^j = m$ for τ_i, where m is the total number of the cores available for real-time data analytics in the system. In this way, we finish a periodic instance of an individual real-time data analysis task as early as possible, while avoiding context switches due to preemptions.

In this paper, we apply non-preemptive uniprocessor EDF scheduling to meet timing constraints of real-time data analysis tasks using the schedulability test for non-preemptive periodic tasks with no idle time insertion [14], because m cores are used as if they are a faster uniprocessor for data and compute intensive real-time data analytics. Specifically, the task set $\Gamma = (\tau_1, \tau_2, ..., \tau_n)$ is schedulable, if the two following necessary and sufficient conditions are met:

Condition 1.

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1 \tag{1}$$

Condition 2. $\forall i,\ 1 < i \leq n$; $\forall L,\ P_1 < L < P_i$:

$$L \geq C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{P_j} \right\rfloor C_j \tag{2}$$

Condition 1 requires the processor is not overloaded. In Condition 2, the tasks in Γ are sorted in non-descending order of periods. The right hand side of the inequality in Condition 2 is a least upper bound on the processor demand realizable in an interval of length L that starts when τ_i's job is scheduled and ends sometime before the deadline of the job. The two conditions are unrelated in that conceiving of both schedulable task sets with the total utilization of 1 and unschedulable task sets having arbitrarily small utilization are possible [14].

If Γ is schedulable subject to Eq 1, Eq 2, and the memory constraint, RTMR schedules the periodic data analysis tasks. Otherwise, it provides feedback to the user so that the user can adjust the task parameters, such as the task periods, or provide faster map() and reduce() functions that may produce approximate results. After an adjustment, the schedulability test is repeated for the modified task set.

We acknowledge that alternative scheduling methods could be applicable. For example, the average execution times rather than the maximum ones can be used, if occasional deadline misses are acceptable to a certain degree. τ_i can use fewer than m cores such that more than one tasks can run together, similar to [15], [16], if the maximum memory access delay and contention for shared resources, e.g., the system bus and memory controller, between concurrent data analysis tasks can be quantified in terms of timing. In a many-core system, the real-time data analysis tasks can be partitioned into multiple sets of the cores using a bin-packing heuristic [13]. In each partition, the tasks statically assigned to the partition can be scheduled using the method described in this paper. However, partitioned scheduling of real-time data analysis tasks is challenging, since bin-packing is NP-complete. A thorough investigation of these issues is beyond the scope of this paper and reserved for future work.

IV. Performance Evaluation

In this section, the micro-benchmarks and system settings used for performance evaluation are described. Also, the experimental results are discussed.

A. Workloads and System Settings

For performance evaluation, the following popular data analytics benchmarks are adapted to model periodic real-time data analysis tasks.

- *Histogram (HG):* A histogram is a fundamental method for a graphical representation of any data distribution. In this paper, we consider image histograms that plot the number of pixels for each tonal value to support fundamental analysis in data-intensive real-time applications, e.g., traffic control or visual surveillance.[4] The input of this periodic task is a large image with 4.7×10^8 pixels per task period. The input data size processed per period is approximately 1.4 GB.
- *Linear Regression (LR):* Linear regression is useful for real-time data analytics. For example, it is applied to predict sensor data values and stock prices. 2.7×10^7 (x, y) points in two dimensional space, totaling 518 MB, are used as the input per task period to model the approximately linear relation between x and y via LR.
- *Matrix Multiplication (MM):* MM is heavily used in autonomous vehicles and many scientific applications. In this paper, MM multiplies two 2048×2048 matrices together per task period. Each input matrix is 16 MB. The output matrix is 16 MB too.
- *K-means clustering (KM):* This is an important data mining algorithm for clustering. For example, it can be used to cluster mobile users based on their locations for real-time location-based services. It partitions ℓ observations into k clusters (usually $\ell \gg k$) such that each observation belongs to the cluster with the nearest mean. The input of the k-means task is 10^7 points in two dimensional space, totaling 77 MB, per task period.

[4]HG is not limited to image data but generally applicable to the other types of data, e.g., sensor readings.

All the benchmarks are reductive; that is, the size of the intermediate/output data of all the benchmarks is not bigger than that of the input data. Among the tested benchmarks, only KM consists of more than one pair of map-reduce phases. Specifically, it is implemented as a series of seven pairs of iterative map and reduce phases. However, it generates no additional intermediate/output data; it only finds new k means and updates the cluster membership of each point according to the new means in each pair of the map and reduce phases. For the tested benchmarks, enough memory is reserved as discussed in Section III.

Our system used for performance evaluation has two AMD Opteron 6272 processors. Each of them has 16 cores running at 2.1 GHz. There is a 48 KB L1 cache and 1 MB L2 cache per core. In addition, there is a 16 MB L3 cache shared among the cores. Out of the 32 cores, one core is dedicated to the real-time scheduler and another core is exclusively used to generate periodic jobs of the real-time data analysis tasks. The remaining 30 cores are used to process the generated real-time data analysis jobs. The system has 32 GB memory. Our prototype is implemented in Linux (kernel 3.5.2) to emulate a real-time data analytics system that can be deployed at, for example, a traffic control or cellular network operation center.

B. Timeliness of Real-Time Data Analysis Tasks

In this section, we profile the maximum (observed) execution times of the tested benchmarks including the computation and data access latency, perform the schedulability analysis of real-time data management tasks offline based on the maximum execution times, and empirically verify whether the deadlines can be met for several sets of real-time data analysis tasks generated using the micro-benchmarks. Specifically, one benchmark is run 20 times offline using randomly generated data. The maximum latency among the 20 runs is used for the schedulability test.

TABLE I
MAXIMUM EXECUTION TIMES IN SECONDS

	m=1	m=2	m=4	m=8	m=16	m=30
HG	2.41s	1.67s	0.88s	0.56s	0.33s	0.2s
LR	1.49s	1.3s	1.18s	0.95s	0.62s	0.37s
MM	19.7s	11.2s	5.9s	3.73s	2.02s	1.11s
KM	10.2s	7.5s	3.72s	3.09s	2.54s	2.36s

Table I show the maximum execution times of the benchmarks derived offline. As the number of cores to process real-time data analysis tasks, m, is increased from 1 to 30, the maximum execution times of the HG, LR, MM, and KM are decreased by over 12, 4.1, 17.7, and 4.3 times, respectively. In HG and MM, load balancing among the cores is straightforward. As a result, the maximum execution time is decreased significantly for the increasing number of the cores used for real-time data analytics. Notably, LR's maximum execution time in Table I decreases substantially only after $m \geq 16$. For $m \leq 8$, the hardware parallelism provided by the employed CPU cores is not enough to considerably speed up LR. On the other hand, the decrease of KM's maximum execution time in Table I becomes marginal when $m \geq 8$. In KM, individual points are often re-clustered and moved among different clusters until clustering is optimized in terms of the distance of each point to the closest mean. Thus, the cluster sizes may vary dynamically depending on the distribution of input points between the consecutive map/reduce phases. As a result, threads may suffer from load imbalance. Thus, using more cores does not necessarily decrease the maximum execution time of KM significantly.

TABLE II
RELATIVE DEADLINES (SECONDS) OF THE TASK SETS ($\Gamma_1 - \Gamma_6$)

	HG	LR	MM	KM
Γ_1	23s	22s	30s	25s
Γ_2	13s	15s	22s	18s
Γ_3	7s	8s	12s	10s
Γ_4	4.5s	5s	7s	6s
Γ_5	3s	4s	5s	6s
Γ_6	2.6s	3s	4s	5s

For performance evaluation, we intend to design a task set with as short deadlines as possible. We have considered the six task sets in Table II where the relative deadlines become shorter from Γ_1 to Γ_6. We consider these task sets to analyze their schedulability for different numbers of cores subject to the two conditions specified in Eq 1 and Eq 2. In each task set, a longer deadline is assigned to a task with the larger maximum execution time. Also, in each task set, we have picked task periods (i.e., relative deadlines) such that the longest period in each task set is no more than twice longer than the shortest period in the task set to model real-time data analysis tasks that need to be executed relatively close to each other. In Γ_6, the tightest possible relative deadlines are picked subject to these constraints in addition to Eq 1 and Eq 2. The maximum execution times and relative deadlines of Γ_6 for 30 cores are shown in the last column and row in Tables I and II, respectively. The maximum total utilization of Γ_6 for 30 cores is set to 1 in Eq 1. Assigning shorter deadlines or bigger data to the tasks in Γ_6 incurs deadline misses.

TABLE III
SCHEDULABILITY OF THE TASK SETS

	m=1	m=2	m=4	m=8	m=16	m=30
Γ_1	yes	yes	yes	yes	yes	yes
Γ_2	no	yes	yes	yes	yes	yes
Γ_3	no	no	yes	yes	yes	yes
Γ_4	no	no	no	yes	yes	yes
Γ_5	no	no	no	no	yes	yes
Γ_6	no	no	no	no	no	yes

Table III shows the results of the schedulability tests for $\Gamma_1 - \Gamma_6$. In Table III, 'yes' means a task set is schedulable for a specific number of cores used to run HG, LR, MM, and KM. All deadlines are met for $\Gamma_1 - \Gamma_6$. We present the performance results for Γ_6 that has the shortest deadlines and, therefore, it is only schedulable when $m = 30$ as shown in Table III. All four periodic benchmark tasks, i.e., HG, LR, KM, and MM tasks in Γ_6, simultaneously release their first jobs at time 0 and continue to generate jobs according to their periods specified

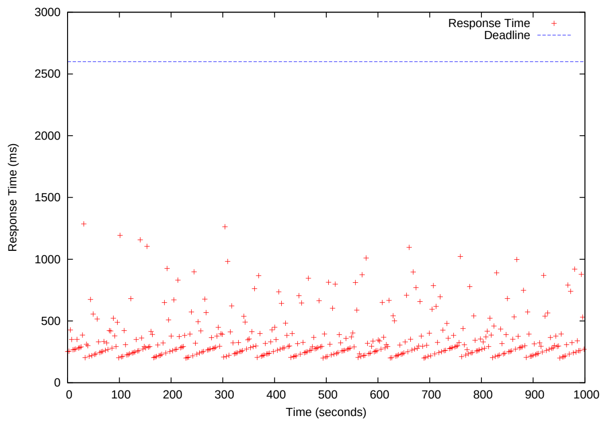

Fig. 5. Response Times of Histogram Jobs (Deadline: 2.6s)

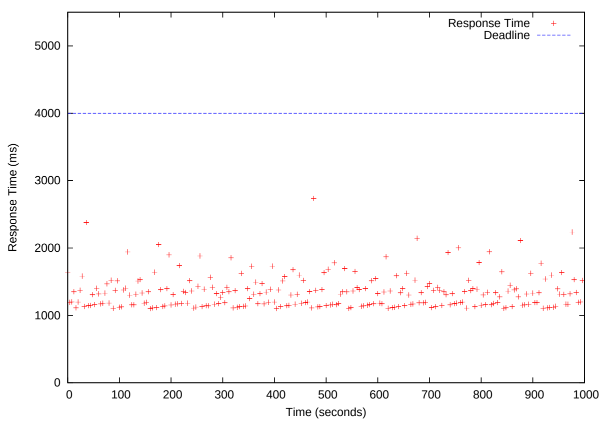

Fig. 7. Response Times of Matrix Multiplication Jobs (Deadline: 4s)

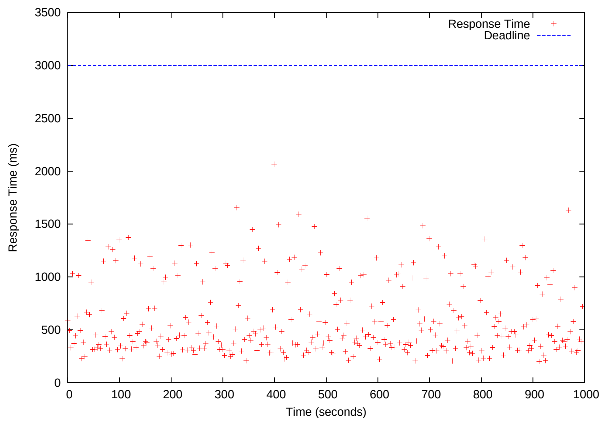

Fig. 6. Response Times of Linear Regression Jobs (Deadline: 3s)

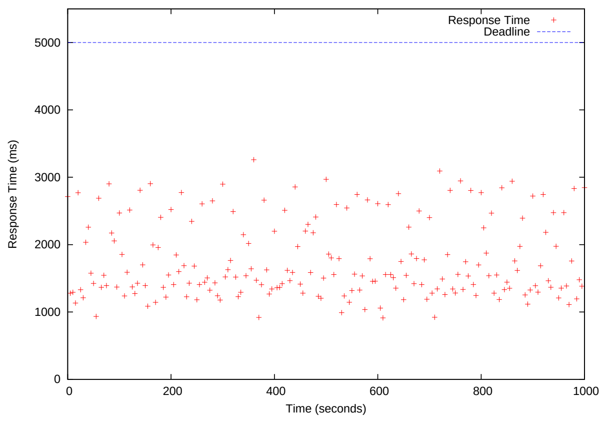

Fig. 8. Response Times of K-means Jobs (Deadline: 5s)

in the last row of Table II for 1000s. As shown in Figures 5 – 8, all deadlines of the periodic real-time data analysis tasks in Γ_6 are met. In total, more than 0.72 TB of data are processed in a 1000s experimental run, which is projected to be over 62 TB/day.

In this paper, Phoenix++ [11] is used as the baseline. It is closest to RTMR in that it is a state-of-the-art in-memory, multi-core map-reduce system unlike the other approaches mainly based on Hadoop (discussed in Section V). However, it has missed most deadlines of Γ_6 for several reasons (although it meets the deadlines for light workloads, such as Γ_1 using 30 cores). First, it is timing agnostic and only supports FIFO scheduling as discussed before. Further, it reads input data from and writes output to secondary storage without supporting input sensor data streaming into memory. Neither does it support in-memory pipelining of intermediate data for iterative tasks. As a result, a single operation to read input data from disk (write output to disk) takes 38ms – 1.35s (71ms – 1.14s) for the tested benchmarks. (More detailed results of Phoenix++ are omitted due to space limitations.)

In Figures 5 – 8, we also observe that the periodic real-time data analysis jobs finish earlier than the deadlines due to the pessimistic real-time scheduling needed to meet the timing constraints. Notably, simply using advanced real-time scheduling techniques that support intra-task parallelism, e.g., [15], [16], does not necessarily enhance the utilization. For example, the best known capacity augmentation bound of any global scheduler for tasks modeled as parallel directed acyclic graphs is 2.618 [16]. Hence, the total utilization of the task set should be no more than $m/2.618$ and the worst-case critical-path length of an arbitrary task τ_i (i.e., the maximum execution time of τ_i for an infinite number of cores) in the task set cannot exceed $1/2.618$ of D_i to meet the deadlines.

Overall, our system design and experimental results serve as proof of concept for real-time big sensor data analytics. Our work presented in this paper opens up many research issues, e.g., more advanced scheduling, load balancing, execution time analysis, and real-time data analysis techniques, to efficiently extract value-added information from large raw sensor data in a timely manner.

V. Related Work

Hadoop [2] is extensively used for big data analytics. A lot of work has been done to enhance Hadoop too [17]. Unfortunately, these approaches based on batch processing of the data stored in the distributed file system neither consider timing constraints nor support periodic in-memory processing of sensor data streams in real-time. HaLoop [18] supports iterative applications in Hadoop; however, it does not support the other key features of RTMR for real-time data analytics.

The problem of meeting real-time deadlines in Hadoop is investigated in [8], [7], [9], [10]. However, these approaches retrofit Hadoop optimized for batch processing of archived data. Neither do they support sensor data streaming or intermediate data pipelining. Hence, they are subject to significant I/O overheads. Also, the deadlines considered by them are tens of minutes long that are inappropriate for real-time sensor data analytics with stringent timing constraints, e.g., traffic control or location-based services. Phoenix++ [11] supports efficient in-memory execution of map/reduce tasks in a multicore system. In this paper, it is significantly extended to support real-time data analytics.

Real-time databases (RTDBs) have been studied extensively to process user transactions using fresh temporal data representing the current real world status. However, sophisticated data analysis based on, for example, machine learning or data mining has rarely been considered in RTDBs. Neither do they provide an easy-to-use parallel programming model, e.g., the map-reduce model [12]. Leading-edge data stream management systems, e.g., Storm [4], S4 [5], Spark Streaming [6], and C-MR [19], support *near* real-time stream data processing. RAMCloud [20] always stores entire data in distributed memory and provides high speed networking to support reads/writes 1000 times faster than disk-based storage. However, they do not consider timing constraints for real-time data analytics. Our approach could be combined with these approaches to handle bigger sensor data in real-time.

In multiprocessor real-time scheduling, it is a common practice to schedule serial tasks concurrently using multiple processors or cores [21]. Novel scheduling algorithms, e.g., [15], [16], are developed to support intra-task parallelism such that a single real-time task can use multiple cores (or processors) at a time. In this way, compute-intensive tasks can meet stringent deadlines that cannot be met otherwise. However, these studies do not consider real-time data analytics issues, e.g., the real-time map-reduce model, data streaming/pipelining, and memory reservation for timely analysis of sensor data.

VI. Conclusions and Future Work

Distilling value-added information from massive sensor data in real-time is desirable yet challenging. Most existing big data management systems, e.g., Hadoop, are timing agnostic and only focus on batch processing of previously stored data rather than dealing with real-time sensor data on the fly. To address the problem, we design a new framework for real-time big data analytics. We have also implemented a prototype system and evaluated its performance using important data analysis benchmarks adapted to model real-time data analytics. In the performance evaluation, our approach can meet the deadlines of the tested real-time data analysis tasks, whereas the state-of-the-art baseline fails to do it. In the future, we will investigate more efficient scheduling and resource management, while providing more advanced real-time data analytics.

Acknowledgment

This work was supported, in part, by NSF grant CNS-1117352.

References

[1] J. Dean and J. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” in *Symposium on Operating Systems Design and Implementation*, 2004.
[2] “Hadoop Project,” http://hadoop.apache.org.
[3] D. Beluke, “Big Data Impacts Data Management: The 5 Vs of Big Data,” http://davebeulke.com/big-data-impacts-data-management-the-five-vs-of-big-data/.
[4] “Apache Storm,” https://storm.apache.org/.
[5] “S4: Distributed Stream Computing Platform,” http://incubator.apache.org/s4/.
[6] “Spark Streaming,” https://spark.apache.org/streaming/.
[7] L. T. X. Phan, Z. Zhang, Q. Zheng, B. T. Loo, and I. Lee, “An Empirical Analysis of Scheduling Techniques for Real-time Cloud-based Data Processing,” in *International Workshop on Service-Oriented Computing and Applications*, 2011.
[8] K. Kc and K. Anyanwu, “Scheduling Hadoop Jobs to Meet Deadlines,” in *International Conference on Cloud Computing Technology and Science*, 2010.
[9] F. Teng, H. Yang, T. Li, Y. Yang, and Z. Li, “Scheduling real-time workflow on MapReduce-based cloud,” in *International Conference on Innovative Computing Technology*, 2013.
[10] 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 Cluster,” in *ICDCS*, 2014.
[11] J. Talbot, R. M. Yoo, and C. Kozyrakis, “Phoenix++: Modular MapReduce for Shared-Memory Systems,” in *International Workshop on MapReduce and its Applications*, 2011.
[12] R. Bird and P. Wadler, *Introduction to Functional Programming*, 2nd ed. Prentice Hall, 1998.
[13] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, “A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms,” in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, J. Y.-T. Leung, Ed. Chapman Hall/CRC Press, 2003.
[14] K. Jeffay, D. Stanat, and C. U. Martel, “On Non-Preemptive Scheduling of Periodic and Sporadic Tasks,” in *Real-Time Systems Symposium*, 1991.
[15] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. D. Gill, “Parallel Real-Time Scheduling of DAGs,” *IEEE Transactions on Parallel Distributed Systems*, vol. 25, no. 12, pp. 3242–3252, 2014.
[16] J. Li, J.-J. Chen, K. Agrawal, C. Lu, C. D. Gill, and A. Saifullah, “Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks,” in *Euromicro Conference on Real-Time Systems*, 2014.
[17] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, “Parallel Data Processing with MapReduce: A Survey,” *SIGMOD Record*, vol. 40, no. 4, pp. 11–20, 2011.
[18] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, “HaLoop: Efficient Iterative Data Processing on Large Clusters,” *PVLDB*, vol. 3, no. 1-2, pp. 285–296, 2010.
[19] N. Backman, K. Pattabiraman, R. Fonseca, and U. Cetintemel, “ C-MR: Continuously Executing MapReduce Workflows on Multi-core Processors,” in *International Workshop on MapReduce and its Applications*, 2012.
[20] “RAMCloud,” https://ramcloud.atlassian.net/wiki/display/RAM/RAMCloud.
[21] R. I. Davis and A. Burns, “A Survey of Hard Real-time Scheduling for Multiprocessor Systems,” *ACM Computing Surveys*, vol. 43, no. 4, 2011.