Redoop Infrastructure for Recurring Big Data Queries*

Chuan Lei, Zhongfang Zhuang, Elke A. Rundensteiner, and Mohamed Y. Eltabakh
Worcester Polytechnic Institute, Worcester, MA USA
{chuanlei,zzhuang,rundenst,meltabakh}@cs.wpi.edu

ABSTRACT

This demonstration presents the Redoop infrastructure, the first full-fledged MapReduce framework with native support for recurring big data queries. Recurring queries, repeatedly being executed for long periods of time over evolving high-volume data, have become a bedrock component in most large-scale data analytic applications. Redoop is a comprehensive extension to Hadoop that pushes the support and optimization of recurring queries into Hadoop's core functionality. While backward compatible with regular MapReduce jobs, Redoop achieves an order of magnitude better performance than Hadoop for recurring workloads. Redoop employs innovative window-aware optimization techniques for such recurring workloads including adaptive window-aware data partitioning, cache-aware task scheduling, and inter-window caching mechanisms. We will demonstrate Redoop's capabilities on a compute cluster against real life workloads including click-stream and sensor data analysis.

1. INTRODUCTION

The availability of large-scale data processing systems [5, 14] has enabled most applications to explore their big data sets using data-intensive analytical tasks that were not possible before. MapReduce [5] provides a simple API for writing user-defined jobs: a user only needs to specify a serial map function and a serial reduce function. MapReduce then executes these functions over massively large datasets so called "big data" in a shared-nothing cluster.

Hadoop [14] is a widely-used platform for such data-intensive applications because of its scalability, flexibility, and fault tolerance. However, as proven by a flurry of research work on Hadoop [11, 9], distributed processing by itself is not enough to achieve high performance. Instead, the system needs to be highly optimized to achieve robust performance for specific classes of query workloads. In this vein, Redoop now is the first to focus on optimizing the type of *recurring workloads* prevalent in many mission-critical applications [10, 13]. A recurring query is defined as an analytical query that periodically executed over large volumes of evolving datasets. In each execution it limits its scope of interest using a sliding window, e.g., processing the last n hours, days, weeks, or even months worth of data depending on the granularity of interest.

We use *news feed updates* as an example of recurring queries in Internet-scale applications. On modern consumer websites, news feed generation is nowadays driven by online systems. Online news services may be generated on a per-member basis based on the member's interactions with other components in the system. For example, a LinkedIn member may receive periodic updates on their profile changes. Computing these updates involves deep analytical processing of large-scale data sets across multiple sources. To generate an update highlighting the company in which most of a member's connections have worked in the past month requires joining the company's data of various profiles. The update may often be delivered to the members by the end of each day or week. Such updates could naturally be expressed by recurring queries over evolving data sources such as members' connections, employers, etc.

Many other data analytics applications from log processing to click stream analysis share similar characteristics: they need to periodically run recurring queries over large volumes of data. Thus even with a relatively small processing window, the amount of data overlapping between consecutive executions can be huge. Therefore, without proper system-level support, e.g., understanding the recurring nature of queries, critical optimization opportunities have been missed.

State-of-the-Art. Several extensions have been proposed to improve Hadoop's performance w.r.t different query types, e.g., SQL-like queries [7, 12], online processing [4], and iterative queries [3]. However, none of these systems support or optimize the recurring big data queries addressed by our Redoop solution [8].

While some studies such as HaLoop [3] and Twister [6] have utilized disk-based caches to improve the performance of Hadoop, their domain of queries, which is the $iterative$ queries, is different from ours. These systems provide special purpose support to identify and then maintain invariant data during subsequent recursions on same static data set. The major difference between the aforementioned systems and Redoop is that we use a well-understood set of principles from window semantics to provide an end-to-end optimization for supporting recurring queries over evolving data sets. Thus our Redoop infrastructure is a general purpose solution that not only offers caching mechanism to avoid redundant computations but also tackles other issues in recurring query processing such as fluctuating inputs and window-aware task scheduling.

Nova [11], closest to our work, supports the convenient specification and processing of incremental workloads on top of Pig. However, Nova acts as a middle-ware layer on top of Hadoop which is treated as a black-box system. It cannot exploit the optimization opportunities offered by Redoop including adaptive data partition-

*This project is supported by NSF grants CNS-305258, IIS-1018443 and IIS-0917017.

Proceedings of the VLDB Endowment, Vol. 7, No. 13

ing, caching of the intermediate data to avoid redundant shuffling, cache-aware task scheduling to utilize cache locality, and adaptivity to the data arrival rate.

In summary, existing MapReduce-like systems fall short in providing system-level optimizations for recurring big data queries. They offer neither caching of intermediate data for reuse, cache-aware task scheduling, nor adaptive processing based on input data rates. Our Redoop [8] infrastructure is the first MapReduce-based technology that offers dedicated support for data-intensive recurring queries. Highlights of Redoop to be showcased include:

• **Redoop recurring query model:** Redoop efficiently handles recurring queries with a wide spectrum of execution granularities through a proactive execution mechanism, whereby it adaptively detects fluctuations in input data between different executions and proactively starts performing partial processing to deliver results.

• **Redoop adaptive data partitioning strategy:** Redoop splits the input data into fine-grained data units (called panes) customized for effective window-centric data consumption. Adaptive partitioning eliminates costs of repeated reading and loading of partially overlapping panes across windows.

• **Redoop window-aware caching:** Redoop creates re-use opportunities across the subsequent execution of recurring queries by caching intermediate data at different stages of a MapReduce job. The caching mechanism significantly reduces I/O costs by avoiding unnecessary re-loading, re-shuffling, and re-computation of the overlapping data. The Redoop scheduler is tuned to maximize the utilization of the available caches and to balance the workload on each node to boost the system performance.

In this demonstration, we show Redoop infrastructure running over real-world workloads with fluctuations including click-stream and sensor data analytics. The demonstration will leverage a 30-node compute cluster hosted in the Computer Science Department at Worcester Polytechnic Institute.

2. REDOOP INFRASTRUCTURE

The sliding window semantics of recurring queries may result in a significant overlap of data between consecutive windows. Thus, we proposed an advanced task execution manager for Redoop to cache input data on local file systems of task nodes. The cached data is efficiently utilized to reduce redundant disk I/O operations at run-time. Redoop introduces an incremental processing model to allow task nodes to asynchronously execute any map or reduce task with incrementally evolving data between two consecutive query recurrences. Redoop adds five new components along with adopting and extending several existing components from Hadoop. Figure 1 illustrates the proposed infrastructure of Redoop as an extension of Hadoop.

1. Window Semantic Analyzer is the optimizer that, given the window constraints of recurring queries, produces a data partition plan. That is, it produces a plan of subdividing input data sources into panes (i.e., separate HDFS files) with optimized granularity that can be most efficiently processed by map and reduce tasks. Such plan can also eliminate unnecessary data re-processing caused by recurring queries.

2. Dynamic Data Packer is the partition executor that implements the instructions encoded in the partition plan produced by the above optimizer. That is, it dynamically splits very large input data partitions into smaller panes. The data packer piggybacks the pane creation step with the loading step, i.e., while a given input file is being loaded into HDFS, the data packer partitions the records to the corresponding panes.

3. Execution Profiler collects the statistics after the completion of each query recurrence, i.e., execution times of previous query recurrences. The profiler then transmits the statistics to the Window Semantic Analyzer such that the pane size can be adjusted in a timely manner during subsequent input partitioning. The above three mentioned Window Semantic Analyzer, Dynamic Data Packer, and Execution Profiler together also determine the Redoop's execution modes to tackle data fluctuations.

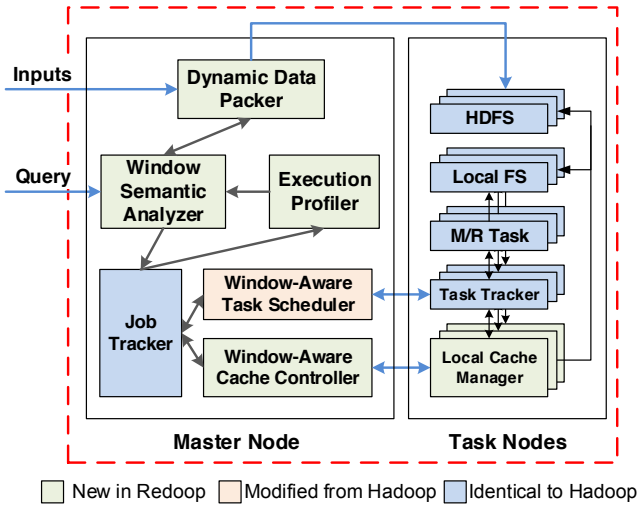


Figure 1: Redoop Infrastructure [8]

4. Local Cache Manager installed on each task node in Redoop maintains the Redoop caches on the node's respective local file system. The Local Cache Manager sends its cache meta-data to the Window-Aware Cache Controller described below along with its heartbeat for global synchronization. The cache manager allows users to provide a purge policy. It is responsible for purging the expired caches according to the prescribed policy and the purge notification received from the master node.

5. Window-Aware Cache Controller is a new module housed on the Redoop master node that maintains window-aware meta-data of reduce input and output data cached on any of the task nodes' local file systems. This controller helps optimize query execution by providing information of window-dependent cache usage for run-time task scheduling decisions.

6. Window-Aware Task Scheduler, an extension of the default Hadoop TaskScheduler, aims to maximally exploits the intermediate caches that reside on the local file system for incremental window-centric processing of input data. It also balances the workloads on each node based on the locality of prior caches. Both exploiting existing caches and keeping the load balanced further both improve the query processing performance.

3. KEY TECHNICAL INNOVATIONS OF REDOOP SOLUTION

This section discusses the key innovations of Redoop to support large-scale data-intensive recurring query processing using MapReduce paradigm.

Adaptivity to Load Fluctuations: Evolving data fluctuating over time requires Redoop to adapt to these changes. Variance of the input data sources (in rate and/or in values) can at times result in temporary load spikes, with the data processing time significantly affected by the duration of the spikes. Worse yet, the cluster resources may not be efficiently utilized and the delayed query results may further slow down other data analytics jobs that depend on the current query execution.

Redoop implements an *adaptive pane-based partitioning* technique to adaptively partition a pane into sub-panes when faced with workload spikes. Clearly, a larger amount of input data will tend to increase the execution time. The core idea is to exploit the statistics collected from the Execution Profiler, i.e., the execution times and the amount of data processed in the previous executions, to adjust the pane size during the subsequent input data partitioning process. This is based on the insight that the input data size is one of the dominant factors determining the execution time of a MapReduce job [9]. Thus, the pane size in Redoop is determined by a series of observations of the job execution over time and the corresponding pane sizes. Our solution is to estimate the future behavior of input data sources based on these observations and then produce the pane-based partitioning plan accordingly.

Cache-Based Processing: The greatest challenge and opportunity in recurring queries is that the inputs across consecutive executions of the same query can significantly overlap. The straightforward approach to avoid the repeated input data processing is to keep the overlapping data in memory before any job finishes processing the data. This is not a desirable approach in our case for two reasons. First, the memory resources may not scale to the huge volume of data. Second, it is vital to retain the fault-tolerance of MapReduce via automatic cache recovery and task re-execution support.

In Redoop, we cache the input data partitions on task nodes' local file system for subsequent reuse to reduce the unnecessary I/O costs. Redoop maintains caches at two stages of a MapReduce job, reduce input and output. Both cached data need not to be loaded, processed or shuffled again with the same mapper across windows. Hence this reduces the processing time for recurring queries. To facilitate caching on local nodes, Redoop maintains additional data structures associated with these caches. Due to data sources being updated periodically, the local file system on task nodes cannot accommodate an unbounded number of historical caches. Thus, we also introduce purging mechanisms to remove the expired caches.

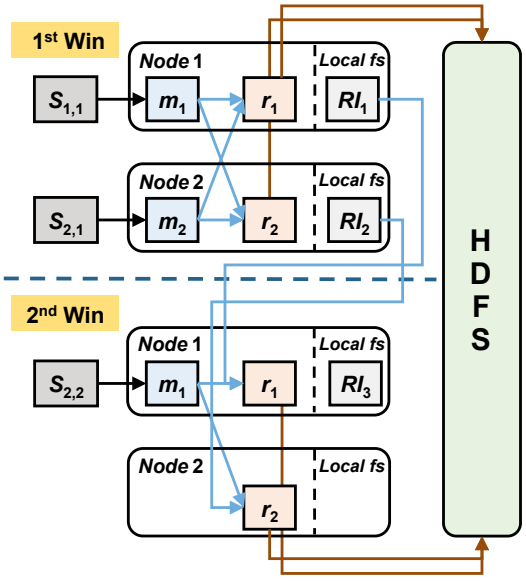


Figure 2: Redoop Infrastructure Workflow

Cache-Aware Scheduling: The default Hadoop task scheduler does not understand the window constraints specified in the recurring queries. Thus the execution of a chain of recurring jobs can be slow, especially if an inappropriate scheduling decision is made. The goal of the Redoop's cache-aware scheduler is to schedule tasks that exploit the window-centric cached partitions as much as possible, reducing redundant work across window panes.

To provide more detail. For example, Figure 2 is a sample schedule for a query joining data sources S_1 and S_2. To improve performance, window-centric partitions $S_{1,1}$ and $S_{2,1}$ from S_1 and S_2 are cached and reused in the recurring query processing. Two task nodes are involved in this job. The scheduling of the first window in Redoop is no different than in Hadoop, except the reducer inputs RI_1 and RI_2 are cached on the local file systems of $Node_1$ and $Node_2$, respectively.

The Redoop's scheduler can take advantage of the cached data RI_1 and RI_2 when it schedules the second execution of the join query. There is no need to load the input data partitions $S_{1,1}$ and $S_{2,1}$, to re-compute their map outputs, nor to communicate them to the reducers. Only the new data partition $S_{2,2}$ needs to be processed. The same reducers r_1 and r_2 combine the new reducer inputs from m_1 and m_2 with RI_1 and RI_2 to compute the results for the second query occurrence.

We refer the interested reader to our technical paper [8] for more details about the Redoop infrastructure and techniques.

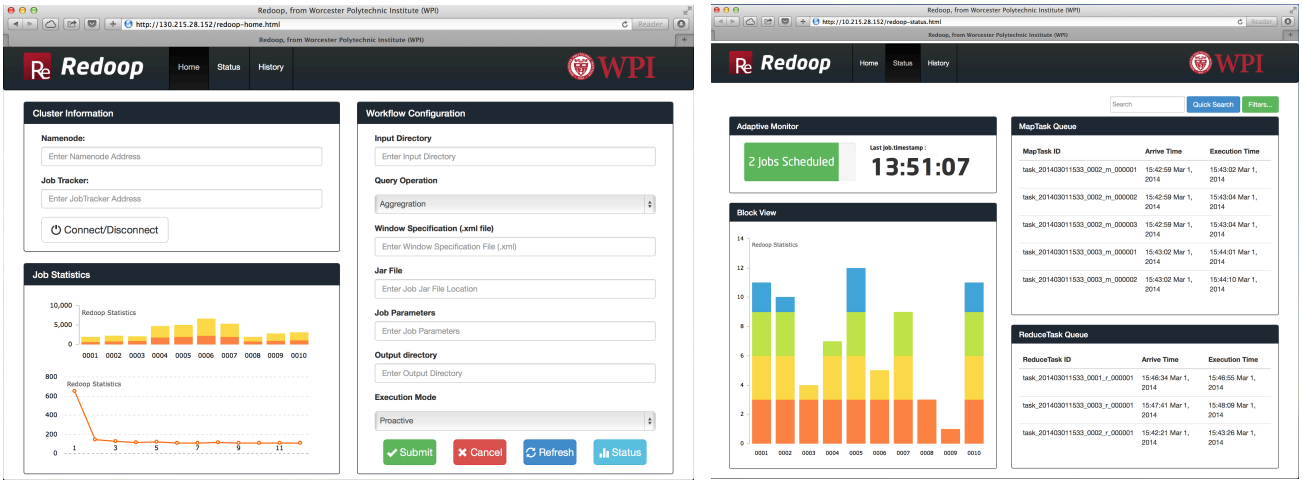
4. DEMONSTRATION DETAILS

The showcases using real use cases including recurring workload configuration and processing. It also includes demonstrations of extreme workflows such as input data fluctuation scenarios to illustrate the effectiveness of Redoop innovations, such as caching, adaptive input pre-processing, robustness of fault tolerance. The use cases are setup such that all jobs complete within 10 minutes, which is the duration of the entire demonstration.

The demonstration has two key components. The first component shows the Redoop implementation in action on real recurring applications, including recurring workload configuration and processing. The second component is an interactive walk-through of the Redoop approach using various workloads that represent input data fluctuation scenarios to illustrate the effectiveness of Redoop innovations, such as caching, adaptive input pre-processing, robustness of fault tolerance. The use cases are setup such that all jobs complete within 10 minutes, which is the duration of the entire demonstration.

Recurring Workload Configuration: We first show how Redoop is used to model and accept recurring queries for different window constraints with real datasets from two domains, namely the WorldCup web click (WCC) [1] and the sensor data collected from a football field (FFG) [2]. The audiences can interactively construct additional recurring workloads for clickstream or trajectory analytics by selecting and adding relevant components such as input data sources and key operations for the recurring application. Redoops UI automatically asks for the relevant parameter values (see Figure 3(a)). The audiences can deploy these workloads with ease on our compute cluster.

Recurring Workload Processing: The recurring workload processing will be demonstrated with both Redoop and the plain Hadoop as a baseline system. The Redoop visual monitor displays metrics including processing time, end-to-end latency, and I/O consumption in Redoop's job report portal (see Figure 3(b)). The audiences can work with different workloads by adjusting the operation type and datasets. Furthermore, a detailed report of query execution progression helps the audience to better understand Redoop's improvements to each processing phase. This display reports on the cost distribution of the query operation across the Shuffle and Reduce phases in a series of consecutive executions.



(a) Redoop Query Configuration

(b) Recurring Queries Performance Monitoring

Figure 3: Redoop Front End GUI

Effectiveness of Window-Aware Caching: The audience can enable or disable the caching functionality during the recurring workload configuration. Performance improvement achieved by the Redoop's pane-based caching and cache-aware execution over the plain Hadoop can easily be observed. Details of Redoop caching strategies can be explored by manipulating parameters such as the number of nodes and the window overlapping ratio. The overlap ratio indicates how much input data partitions are the same between two consecutive query executions. From the impact on the execution time of a recurring application, the audience can assess the scalability and efficiency of Redoop pane-based caching.

Effectiveness of Adaptive Pane-Based Partitioning: To demonstrate the robustness our adaptive pane-based partitioning technique, we work with recurring workloads with fluctuating arrival rates of input data during each execution. We then execute the recurring query over the workload with and without the adaptive pane-based partitioning technique. Redoop's graphical views (see Figure 3(b)) allow for a quick identification of load spikes and the degree of data fluctuation. This reveals that input data fluctuation translate into system overloads. Redoop's analysis screen visualizes the execution times of all map and reduce tasks along with their assigned workload (i.e., number of panes). This illustrates that Redoop keeps similar amount of workload, namely map and reduce tasks, on each node when employing adaptive technique (Sec. 3). Moreover, the frequency of job executions along with the amount of input data that each job actually consumes is displayed.

Robustness of Fault Tolerance: We will demonstrate fail scenarios by removing caches from task nodes. The audiences will learn how our Redoop system can seamlessly recover from cache failure when the cached data is lost. The cached intermediate data of the running query in the background will be removed via our console. The Redoop's query monitor automatically detects the cache failure and shows how Redoop copes with such failure w.r.t the overall processing time. This fault tolerance focussed demonstration will let the audiences assess the robustness of Redoop.

5. CONCLUSION

In summary, this demonstration presents the key innovations of the Redoop infrastructure, a novel distributed system that optimizes the recurring big data queries. We show Redoop's capabilities on a compute cluster against real life workloads including click-stream and sensor data analysis.

6. REFERENCES

[1] 1998 world cup. http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[2] Soccer - real time tracking system. http://www.iis.fraunhofer.de/en/bf/ln/referenzprojekte/redfir.html.

[3] Y. Bu, B. Howe, M. Balazinska, and others. Haloop: Efficient iterative data processing on large clusters. *PVLDB*, 3(1):285–296, 2010.

[4] T. Condie, N. Conway, P. Alvaro, et al. Mapreduce online. In *NSDI*, pages 313–328, 2010.

[5] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, pages 137–150, 2004.

[6] J. Ekanayake, H. Li, B. Zhang, et al. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818, 2010.

[7] Hive. Hive. http://hadoop.apache.org/hive.

[8] C. Lei, E. A. Rundensteiner, and M. Y. Eltabakh. Redoop: Supporting recurring queries in hadoop. In *EDBT*, pages 25–36, 2014.

[9] B. Li, E. Mazur, Y. Diao, et al. A platform for scalable one-pass analytics using mapreduce. In *SIGMOD*, pages 985–996, 2011.

[10] D. Logothetis, C. Trezzo, K. C. Webb, et al. In-situ mapreduce for log processing. In *USENIXATC*, pages 9–9, 2011.

[11] C. Olston, G. Chiou, L. Chitnis, et al. Nova: continuous pig/hadoop workflows. In *SIGMOD*, pages 1081–1090, 2011.

[12] *Pig*. http://hadoop.apache.org/pig.

[13] R. Sumbaly, J. Kreps, and S. Shah. The big data ecosystem at linkedin. In *SIGMOD*, pages 1125–1134, 2013.

[14] The Apache Software Foundation. Hadoop. http://hadoop.apache.org.