2013 IEEE Ninth World Congress on Services

# Towards Real-Time Analytics in the Cloud

Amr Osman
Computer Science Department
*German University in Cairo*
New Cairo, Egypt
amr.salaheldin@guc.edu.eg

Mohamed El-Refaey
Middle East Mobile Innovation Center
*Intel Labs*
Cairo, Egypt
mohamed.elrefaey@intel.com

Ayman Elnaggar
Computer Science Department
*German University in Cairo*
New Cairo, Egypt
ayman.elnaggar@guc.edu.eg

***Abstract*—The data explosion and the tremendous growth in the volume of data generated from various IT services places an enormous demand on harnessing and smartly analyzing the generated data and enterprise contents. According to recent studies, it is predicted that the volume of such data will become 26 fold in the next five years. While there might be some existing technologies to support this, industry is frantically exploring new models that lead to more efficient and higher performance solutions. With the aid of cloud computing and high performance analytics such as scalable-parallel machine learning, big data could be the fuel to a smarter cloud-powered IT world. Through our work, we provide a state-of-the-art review of high-performance advanced cloud analytics in the literature in attempt to find the ideal real-time platform for distributed analytic computations.**

***Index Terms*—Cloud Computing; Real-time cloud analytics; Stream Computing; Map-reduce; Distributed Machine learning; Hadoop; Big data**

## I. Introduction

Cloud computing has evolved into one of the most commonly used and wide-spread technologies nowadays. The recent advancements in utility computing and resource virtualization has enabled numerous businesses to exploit the power of large-scale super computing without the need to invest into expensive operational infrastructure costs (CapEx to OpEx). Having commodotized distributed computing, various use-cases for in-cloud operations are constantly emerging on a daily basis such as backup, hosting, processing and storage offloading, medical analytics[1] as well as various large-scale computations. Even though the cloud is often seen as an option rather than a necessity, the rise of big data will certainly force the transition into cloud and distributed computing. According to current numbers, it is predicted that the volume of digital data will double in size every two years to reach 8 ZettaBytes by 2015[2]. Such explosion in user-generated and system-generated data demands effective analysis, representation and categorization in order to extract valuable and relevant information that could potentially improve the vast majority of IT organizations.

While NIST classifies cloud computing from a service-oriented point of view[3]. Different clouds are not just utility-based as in the SaaS, PaaS and IaaS service models where pooled resources can be efficiently provisioned, elastically scaled and provided to the user on-demand. Data clouds[4] is just the other inseparable other side of the coin. By being data-centric instead of service-and-user-oriented, distributed high-performance computation on large amounts of data is one of the key features that characterize such clouds. The main objective is to aggregate massive amounts of data, divide it into smaller chunks and then distribute processing and storage tasks across nodes within some given cluster(s) in the most efficient, reliable, scalable and fault-tolerant way. Therefore, data clouds are the ideal platform for processing and analyzing big data.

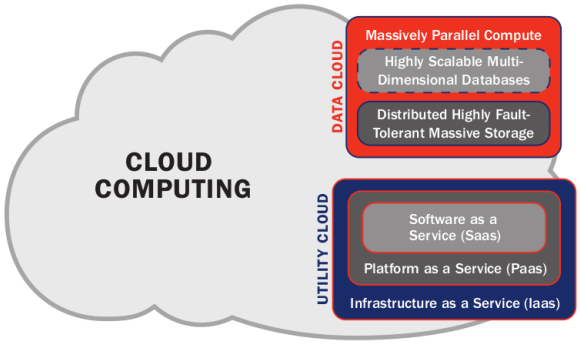


Figure 1. Utility Clouds vs Data Clouds[4]

Even though various approaches have been suggested and introduced to address the problem of analyzing big data in the cloud, achieving high performance, better parallelism and real-time efficiency is still a stumbling block due to the composite and unstructured nature of big data as well as the complexity of analytic algorithms. Therefore throughout this paper, we attempt to review and explore various proposed solutions to implement efficient, high performance parallel analytics inside data clouds. We first study the nature of big data and different types of analytics and state the main challenges. Then, we highlight some state-of-the-art various technologies to efficiently store and perform distributed computation in search for the ideal platform. We further expand to cover various implementations of cloud analytics and approaches to parallelize existing analytic algorithms. Finally, we summarize by providing a comparative review of our findings and lessons-learned to consider during future research.

978-0-7695-5024-4/13 $26.00 © 2013 IEEE
DOI 10.1109/SERVICES.2013.36



IEEE computer society

## II. Overview

Before we discuss the various cloud analytics approaches, we first shed lights on the nature of big data, the factors affecting it and why it is difficult to process. We also need to define and classify cloud analytics beside the fitness of various classes of analytics & machine learning for various scenarios.

### A. Big Data

In contrast to popular allegations, big data isn't just about the data volume. Unlike traditional structured data, it is usually unstructured and varies in types, sizes and ways of classification and storage. A recent case study[5] investigated the nature of big data in different real-world IT businesses and showed that big data depends on 3 different aspects: velocity, volume and variety. These three dimensions are co-related and directly influence the methodologies used to store and process the data. Consequently, affecting the different analytics techniques that could be used as well as the host data cloud technical specifications such the distributed storage system, the distributed database and the cluster management system.

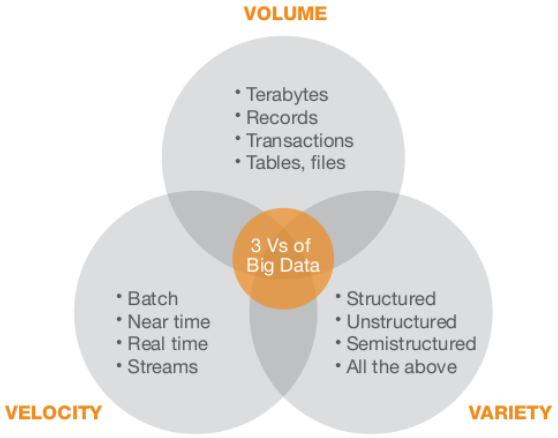


Figure 2. The three Vs of Big Data[5]

The volume dimension does not only define the size of the data. In fact, it directly depends how the data and the associated meta data is stored. Data could be stored as big files, records of multiple file blocks or as tables. As the volume grows, the processing and IO load on the cluster increases which dramatically impacts the performance. The variety factor also indicates the structure of the data since, it could be a mixture of structured and unstructured data. Therefore, it could be highly multi-dimensional and sparsely distributed. The more structured the data is, the easier it becomes to perform analytics and infer relations between the various datasets which are fed into the analytic engine since pre-processing via categorization, labeling and clustering could be skipped all together or reduced. Moreover, the velocity factor is two-pronged: The rate at which data is generated or input to the system contributes to how often it needs to be processed in order to update the previous results. Time-sensitive applications that require real-time response & processing are therefore, hard to support as the three factors of big data increase.

### B. Cloud Analytics and machine learning

Performance also depends on the time-space complexity and degree of parallelism of the undertaken analytic algorithms. Nonetheless, analytics differ according to the goals and expected outcome with respect to how advanced and detailed the resultant models should be. It's a trade-off between fine-grained and coarse-grained tuning. As stated in the previous section, the nature of data and problem plays an important role in using/discovering the appropriate algorithm(s).

Traditional analytics such as Business Intelligence, Operation Research and Data Mining are no longer sufficient enough[6] to harvest value from big data and automate decision making or provide a feedback loop into systems for future improvement and self-tuning. Data scientists and decision makers often have to analyze the data either visually or numerically in order to get an insight. Advanced analytics however, doesn't stop at this point. The overall goal is to exploit and infer complex relationships between data and quantitatively measure the amounts and quality of data. This usually involves machine learning and statistical analysis to make solid mathematical models and abstractions that could be used to predict future behavior or even optimize for multiple goals algorithmically.

Advanced analytics is briefly the discipline of scaling and parallelizing machine learning algorithms on big data inside the cloud in order to detect patterns, classify data and infer statistical relationships for effective data mining use. Learning algorithms could be divided into three main categories:

- **Supervised Learning**: Input training data is usually labeled and accompanied with an expected output per item. Usually, one or more scoring functions are used to evaluate the inferred hypothesis functions in order to obtain the fittest hypothesis function from the hypothesis space. Examples: Bayesian networks, Decision trees, K nearest neighbor.
- **Unsupervised Learning**: Unlike supervised learning, the input is unlabeled and no sample expected outputs exist to evaluate the learned functions. Examples: k-means clustering, Neural networks.
- **Reinforcement Learning**: Is the problem of teaching a decision agent on how to perform actions in a specific environment in order to meet a set of end goals according to a reward function. No input or output data is provided. The agent gains knowledge by performing actions using one or more transition functions and observing the environment's response through a series of episodes. Examples: Genetic algorithms, Simulation-based learning.

While machine learning is the main scientific foundation of cloud analytics, Raden[7] differentiates between three main categories of advanced analytics: Descriptive analytics, Predictive analytics and Optimization analytics.

Table I
DESCRIPTIVE, PREDICTIVE AND OPTIMIZATION ANALYTICS

| Descriptive | Predictive | Optimization |
|---|---|---|
| Usually associated with data mining and segmentation. Employs classification and<br><br>categorization of data leading to new associations and probability analysis Yields information about<br><br>past data patterns: What happened, how many, how often and where. | Application of complex math, machine learning and statistics to detect patterns and anomalies<br><br>The outcome of descriptive analytics could be combined in order to understand causes and relations which could be used to statistically predict the future.<br><br>Provides information on what will happen, what could happen and what actions are needed. | Usually concerned with optimized and efficient decision making in terms of the learned patterns and predicted probabilities.<br><br>Describe the best possible outcome given a set of outcomes (Resulting from descriptive and predictive analytics) using probabilistic and stochastic methods. e.g. Monte Carlo, Beyesian trees |

Likewise, another paper[8], divides analytic & machine learning models into 6 main cases:

- **Statistical:** Apply probability analysis and analyze information entropy.
- **Association:** How could an item be related to others in the data set?
- **Clustering:** Group several items by likelihood and similarity.
- **Binary Predictions:** True and false decisions.
- **Number-in-range predictions:** Use of regression to determine outliers and predict variables.
- **Selections:** Planning and optimizing for best possible decision or solution after exploring other possibilities.

The transformation into a complete analytic system consisting of highly unstructured big data could therefore be visualized as applying supervised and unsupervised learning techniques throughout a pipeline consisting of descriptive, predictive and optimization analytics consecutively. The machine learning algorithms used are dependent on the analytic models as mentioned above. With that into consideration, the key challenge is to provide parallel and scalable implementations of supervised and unsupervised learning that would run efficiently in a distributed environment.

## III. LITERATURE REVIEW

Through this section we research and compare current approaches towards big data parallel processing and distributed computation inside the cloud using Map-reduce. We explore the current distributed implementations for machine learning and data mining and review the outcomes of each experiment with respect to key factors such as scalability, fault-tolerance and performance. We hope to exploit real-time computation and online implementations of analytic algorithms that are suitable for time-critical applications.

### A. Map Reduce

Map-reduce[9] is a programming model that was recently developed by Google in order to facilitate parallel programing and distributed execution on large clusters. It is based on a no single point of failure architecture that is guaranteed by the underlying distributed file system (GFS[10]) which divides the data into smaller chunks, stores it and safely replicates it across all nodes. The idea behind MR (Map Reduce) is to provide abstraction from the underlying hardware and eliminate the complexities of typical parallel programming models such as MPI (Message Passing Interface). This is achieved by introducing two key functions for processing the data: A *map* function and a *reduce* function which execute back to back in the pipeline. We mathematically formalize these two functions as follows: the map function $m$ is represented as $\langle k, v\rangle \overset{m}{\longmapsto} \bigcup_{i=1}^{n} \{\langle k'_i, v'_i\rangle\}$ whereas the reduce function $r$ which further processes the map function's output is represented as $\left\langle k'_i, \overrightarrow{V''}\right\rangle \overset{r}{\longmapsto} \overrightarrow{V'''}$ constrained by:

$$\forall_{\langle k'_i, v'_i\rangle \in m(\langle k,v\rangle)} \exists_{\langle k'_i, v'_j\rangle} \left[\{v'_i, v'_j\} \setminus V'' = \phi\right] \tag{1}$$

$$\forall_{\langle k'_i, v'_i\rangle, \langle k'_k, v'_k\rangle \in m(\langle k,v\rangle)} k'_i \neq k'_k \Rightarrow v'_k \notin V'' \tag{2}$$

$$|V'''| \leq |V''| \tag{3}$$

where $k, v$ indicate an arbitrary input key along with its corresponding value from the input data respectively and $V'', V'''$ are both lists of values in the form: $\{v_1, v_2, v_3, .....v_n\}$. The reduce function is preceded by a grouping function which groups the intermediate output values from $m(\langle k, v\rangle)$ by key. This can be shown in constraints (1) and (2). The reduce function then performs some arbitrary computation and reduces the input set of values $V''$ into a smaller set $V'''$ as shown by constraint (3).

The flow of execution is simple. All nodes fork a local copy of the user program. A master node then assigns some map tasks to a group of worker nodes and some reduce tasks to another group of workers. The map function executes concurrently across all mappers and the output is written locally. Only then, the intermediate outputs are aggregated, grouped according to the key value and then the reduce functions start executing in parallel across reducer nodes which finally reduce the output to a smaller set of values that are written to the global distributed file system. It follows that MR suffers a major performance bottleneck which is that the reduce phase does not begin until the map phase is complete. Moreover, it is only suited for batch processing and favors reliability, scalability and throughput over latency and execution-time[11], [12]. Consider the scenario when reduce nodes perform remote reads over the network before execution. Another scenario in GFS (Google File System) and

HDFS (Hadoop Distributed File System) when duplication is enforced by the primary replica before providing a response. Therefore, real-time processing is not viable as execution stops when all reducers finish execution and write their output to GFS. In addition, the input data is immutable and stored as chunks which are passed as input to the user program at run-time. Meaning of which, performing computation on live dynamic data that is frequently changing is not supported. This is also a limitation of the underlying file system besides, the assumption of rare random read/write operations[10], [13].

### *B. Optimizations for MR*

Various optimizations have been proposed in order to improve hadoop[13] - the open-source implementation of Google MR - and tune it for performance and continuous processing suited for the needs of big data analytics and data-centric computation (i.e mining and machine learning). We classify these optimization techniques as follows:

- Job Scheduling & MR tasks distribution optimizations.
- Networking & I/O optimizations.
- Continuous cascaded MR work-flows.
- Optimized data-queries-oriented approach.
- Real-time optimization.

Map-reduce++[14] suggested optimizing the overall response time by using a SJF-like (Shortest Job First) scheduling strategy by pre-calculating each task's time cost and executing the smaller tasks for each job first. Another paper[15] follows a similar path by introducing a scheduler which allows dynamic resource provisioning according to user-defined performance goals by setting QoS priorities to different jobs which was done by estimating the number of time slots that could be allocated in-parallel to each job. Such method could optimize performance dramatically when running multiple jobs at the same time or enforcing time-constraints on different near-time analytics. Similarly, an algorithm for optimizing hadoop configuration and hideous parameters that affect different resources: CPU, Disk, Network per MR application has been proposed[16]. It follows a signature based approach where the currently executing application's pre-computed signature is compared to a set of already-computed signatures of resource-utilization statistics for which the optimum hadoop parameters were estimated.

Starfish[17], [18] smartly tackles the same collective problems of the previous approaches and adds automated online real-time optimization capability without the need for any user-interaction. It provides job-level tuning by profiling the behavior of various tasks per job as well as the configuration parameters with respect to resource utilization using a technique similar to [16] as discussed above. It also introduces workflow-aware scheduling where the relations between multiple jobs modeled as a DAG (Directed Acyclic Graph) which operate on the file system's data are examined to ensure maximum data-locality by moving the computation to data and reduce data-transfer overhead. The computation time of various jobs with respect to the job-level configurations is also speculated using a search tree to ensure best job-scheduling scenarios. Finally, workload-level tuning is also provided to ensure better provisioning, utilization and elasticity of the cluster taking into consideration the job-level and workflow-level optimizations undertaken. Word-count and Tera-sort benchmarks[9] yielded 1.92x and 1.47x speedup respectively versus the recommended configuration parameters.

Similar approaches([19], [20]) follow the same footsteps and attempt to solve the same problems of elastic provisioning and resource-aware scheduling of MR jobs and sub-jobs targeting the findings of [21], [22] which thoroughly pinpointed key factors that affect hadoop's performance. However, two solutions stand out by targeting different problems. Facebook proposed various optimizations to the hadoop file system (HDFS) for real-time low-latency usage[11] by implementing RPC (Remote Procedure Call) timeouts instead of waiting for longer TCP socket timeouts, supporting explicit mutation-lease revocation to speedup handing locks to writers, supporting reads from local replicas, supporting concurrent readers and writers and eliminating the need to wait acknowledgements when performing flush operations. Microsoft takes a different approach[23] and introduces the concept of filters to reduce the amount of data transfered from storage nodes to computation nodes when initiating a MR program. This is done through a pre-processing stage where all the execution flows that lead to an intermediate output from the map function are examined via byte-code static analysis. Rows or lines in the input data that do not contribute to an output are eliminated. Similarly, columns or data attributes per row that are not needed by the mapper are also eliminated. By generating and injecting the filters into the original MR program, no special modifications are needed for hadoop. Such process yielded results as high as 62%, 25% reduction in run-time and data-transfer respectively for some benchmark jobs.

Hive[24] and Pig[25] take a different route to facilitate data mining and aggregation. By exploiting the fact that data is usually aggregated and structured using a query-like manner, these two projects implement a SQL-like query language to structure, retrieve, store and aggregate data. While hive supports storing the data in the form of distributed tables on the file system, both projects can be used in conjunction with distributed NoSQL massive databases such as HBase[26] - The open source counter part of Google's BigTable[27] - and Cassanda[28]. Hive supports a subset of the SQL language which can be safely parallelized and applies basic SQL optimizations such as column pruning and predicate push-down to filter data as early as possible before the data-size explodes up the predicate trees. It also adds special optimizations to improve data-locality and reduce data-transfer overhead such as pruning unnecessary files from partitions on the file system and buffering small tables in the distributed main memory of worker nodes for faster access. Pig provides a special language to easily describe the flow of data and how it's joined during processing. Both hive and pig transform the queries into a sequential dependency-satisfied pipeline of efficient MR

jobs that are scheduled for execution on the cluster. Such approach, while focused on query optimization could increase performance compared to native MR applications that are poorly written without considering the nature of the data. Moreover being data-oriented, it is more intuitive to describe the data flow and query the data on a high-level instead of traditionally writing MR programs.

We argue that the above solutions are still not suitable for real-time cloud computations and suffer multiple draw backs. All the above workarounds use pre-processing for estimating resources' consumption, completion-time and in case of Microsoft's "Rhea", static code analysis. Typically, the jobs are run on a small subset of the input data in order to speculate such measures. Even though pre-processing takes much less time than actual jobs, time-sensitive applications cannot afford such cost. In the same context, processing on a continuous flow of live and dynamic data was still not addressed by any of the above solutions. Even though data-locality and resource-awareness were thoroughly addressed, only batch computations can be performed by design. Moreover, continuous iterative processing of map-reduce tasks is vital for machine learning algorithms such as k-means clustering[29] where the output of the current iteration depends on the previous iteration during recalculating the centroids. Another example would be Genetic Algorithms (GA)[30] where the next population depends on individuals of the previous generation which are used for mutation and crossover.

### C. Continuous MR work-flow

Addressing the limitations of the previous optimizations, Twister[31] and HaLoop[32] extend MR by adding iterative support for MR tasks within each job. Map and reduce tasks within a job are executed repetitively until a specified stopping condition is met. All intermediate data per MR task per iteration are cached in the distributed main memory of the worker nodes or the local disks for faster low-latency access during next iterations. In addition, tasks that operate on some partitions are scheduled on the respective nodes that contain such partitions in order to minimize data transfer and fully exploit data locality. HaLoop used k-means clustering in order to compute the page rank of two large data sets (46GB and 54GB) and reduced the execution time from ~42s to ~7s compared to Hadoop. It was also observed that only 4% of the intermediate data was shuffled during MR. Twister implemented two algorithms for gene processing: Pairwise distance calculation and multi-dimensional scaling. Results proved that twister maintained 70% and 80% parallel-efficiency respectively for each algorithm.

While the previous implementations could be set for continuous data processing, they are still impacted by some of the previous limitations discussed in the previous section. Operating on dynamic and frequently changing data is still not possible. In addition, computation is not triggered by new data. Similarly, the response time is still high (ranged from several seconds to minutes) and the model is focused on batch processing instead of a real-time data-driven approach. Nonetheless, there is still a pipeline stall between map tasks and reduce tasks.

One approach[33], [34] attempts to solve this problem by eliminating the stall between map and reduce tasks and providing online continuous aggregation at the expense of data accuracy. This is done by pipelining tasks within each job as well as inter-jobs and returning non-final outputs while in the process of computation therefore, minimizing the response time dramatically. Reducers can start computation and consuming the data as soon as (while) the mappers start producing outputs (sensing for new data via polling) and also commutatively aggregate the result and yield it directly in stead of waiting for the final result. Using different approximation techniques, the non-finalized outputs could be "good-enough" compared to the long-awaited final output. A WordCount experiment yielded 25% shorter completion time. However, one major drawback impacted the response time which is the fact that mappers and reducers compete for resources since they now run together and not in two phases as with the traditional MR approach.

Nova[35] attempts to support continuous incremental and non-incremental processing using Pig where new data could be processed independent of the previous data or with respect to some kept state of the old data beside, the traditional batch processing approach where computation is done on all the data from scratch. Some optimizations to improve data locality inspired by the previous approaches are also supported: Jobs that operate on same data are pipelined and co-scheduled to minimize data-reading overhead. Query optimization techniques supported by pig are also adapted since Nova is built on top of Pig. Moreover, garbage collection and data compression minimizes the merge step's overhead as well as the data size stored/transfered. Nova also introduced various computation triggers to address one of the limitations of the other MR optimization techniques. Namely, data-based triggers could toggle computation on arrival of new data; time-based triggers to set the computation frequency and finally, cascade triggers that could specify inter-job dependencies or trigger one or more jobs once a different job reaches some computation state. It was admitted that Nova is still limited by hadoop and Pig that are only suited for batch computations as we argued earlier. It only tries provide lazy triggered processing targeting maximum throughput at the cost of latency.

Spark[36] exploits the concept of distributed shared memory computation and eliminates the significant HDD I/O latencies by introducing in-memory distributed partitions: RDD (Resilient Distributed Datasets)[37]. Such partitions are kept persistently in the memory of the cluster nodes and are recoverable through "lineage" information which is simply the sequence of steps taken to construct an RDD from the input data. It targets iterative applications where the intermediate data is reused frequently across multiple computations. In case the memory capacity does not fit, RDDs are flushed

to the HDDs according to their user-defined persistence-priorities. The developer also controls how to create and what to store in RDDs during implementation beside the main MR job. Map and Reduce tasks are further scheduled nearer to where the corresponding RDDs are for maximum data-locality. Two famous machine learning algorithms were tested: Logistic regression and K-means clustering. Both were tested for 10 iterations on a 100GB-large dataset on 100 machines. Spark was proven 25.3x and 3.2x faster than hadoop for both algorithms respectively.

Chukwa[38] takes a different turn and provides a real-time log aggregation and monitoring system by exploiting the fact that mappers can be seen as data generators (log agents) and reducers are data collectors. The key objective was to disseminate various logs at a sustainable high rate with minimal impact on resource consumption while providing fault-tolerance and scalability. Two options for data delivery were provided: A reliable one which writes and replicates the data to the global file system and a faster non-reliable method where data is streamed through the network and stored in local hard disks to provide the shortest response time possible. An experiment where the MR ratio was 200:1 showed that chukwa achieved the maximum possible throughput that was limited by the underlying file system and HDD rates @ 30 MB/s. A 1:1 ratio is maintained between the data rate per agent and the data collector until the limit is reached showing that the data is delivered with minimum latency at an extreme rate. Furthermore, another experiment showed that the overall rate scales linearly up to 200MB/s (limited by the number of nodes used) as the number of collectors increases. With such high-rate, stream processing on the delivered data is essential to minimize latency and couple data aggregation with actual processing and live analytics.

### *D. Distributed real-time stream computing*

To fill this gap, many solutions have evolved to leverage real-time computation on big data streams. Yahoo S4[39], Aurora[40] and Twitter Storm[41] are 3 similar solutions that tackle the streaming big data problem. While there are some minor technical and naming differences, their approaches can be narrowed down to one common approach. Data is fed into the system as an infinite sequence of tuples which are transfered to processing nodes. The processing nodes can then consume the data tuples and publish some results or produce more data streams to be processed and/or aggregated by other nodes. In addition, intermediate streams' data tuples are usually buffered in the distributed memory of the worker nodes for minimum latency during data access.

S4 relies on event delegation to relay computation-triggering events between processing nodes. When nodes receive new events, they start consuming the incoming stream and emitting one or more intermediate data streams in a MR fashion. HStreaming[42] also implements a similar functionality on top of Pig and Hadoop. An experiment to calculate the click-through rate on ads coupled with live search traffic was run for a period of two weeks where 250,000 users make one million searches per day. The system could sustain 1600 events per second in such case. The peak rate observed reached 26.7Mbps. Similarly, Google BigQuery[43] provides real-time querying on structured data. However, it assumes that the data is structured & stored on the underlying file system and does not process streams as they are entering the cloud. Twitter storm on the other hand could process a million tuples of data per second per node as benchmarked on the live twitter social network traffic.

Aurora tops other approaches by adding query optimization techniques and supporting real-time continuous queries on the data. QoS (Quality Of Service) is also specified for each application to ensure optimum scheduling while maintaining dynamic load balancing according to the incoming data stream and the resource-utilization statistics. Discretized streams[44], [45] is an extension to bring stream-processing to Spark which was analyzed in the section above. The in-memory storage/computation combo with the aid of RDD could transform this approach into the ideal stream processing framework. Streaming computations are performed deterministically through discrete time intervals to maintain constant sub-second latencies and mitigate stateful dependencies on the tuples' order of arrival across different streams. The streams are typically tuples consisting of RDD objects to be processed via one or more MR tasks. D-Streams further optimized Spark for streaming by allowing Asynchronous network I/O and pipelining tasks within each discrete time window which consume or depend on the same in-memory RDDs. It is argued that even though fixed computation intervals introduce latency, such technique provides faster deterministic fault tolerance and RDD recovery which is assumed to be frequent compared to record-at-a-time systems. A thorough benchmark was run to compare Spark-Streaming to S4 and Twitter Storm. S4 was observed to be 10x slower versus Spark and Storm which achieved similar response times. However for smaller record sizes, Spark beat Storm by a factor of 2x on a cluster of 30 nodes and achieved higher throughput due to the batched computation over fixed intervals property.

To improve Twitter-Storm's throughput and respond to the surprising findings of Spark streaming's last benchmark, Aeolus[46] introduces timely batches similar to Spark streaming and DAG plan optimization for Storm. It optimizes the batch-sizes and degree of parallelism for the nodes in order to maximize throughput and balance between computation time per batch and output rate which is mathematically related to the response time. However, such optimization is not adaptive such that dynamic optimization is suggested for future work. Additionally, no benchmark results are available yet.

### *E. Distributed machine learning*

Having reviewed the MR computation frameworks which have matured towards real-time data-driven processing, scalable and parallel machine learning algorithms to perform online analytics are the next challenge. The Apache mahout

project[47] provides a set of machine learning algorithms that were parallelized and tuned for scalability and MR execution. However, all algorithms as well as Mahout itself were designed under the old assumption of batch processing on traditional hadoop clusters with no support for iterative processing or online stream processing techniques as we deeply discussed through our work. Nevertheless, the provided parallelization techniques could be tweaked and tuned for online incremental processing.

Presto[48] facilitates continuous analytics on distributed in-memory arrays via the R statistical programming language. Continuity is maintained through events and data triggers. On average of various analytic algorithms, presto was 37x faster than hadoop. Another approach used Spark Streaming to implement an online Expectation Maximization algorithm to infer travel times in GPS data[49]. Following the same context, other online machine learning algorithms were developed such as [50], [51], [52], [53], [54], [55]. An intensive research[56] also deeply discussed the theory behind online machine learning and its implications. Other proposals[57], [58], [59], [60], [61] aim to optimize for unbound big data and scalability during MR parallel execution where the iterative algorithm is decomposed into a series of MR jobs that are scheduled on the cluster. It follows that online implementations of machine learning on big data are more complicated and harder to parallelize. The literature lacks such implementations which are crucial for advanced online analytics.

## IV. Conclusion and Future work

Stream computing and low-latency map reduce clusters are key enablers to perform distributed real-time analytics on live and dynamic big data streams. The transition from batch processing to stream processing inside the data cloud fits various time-sensitive applications and real-world use cases such as real-time search engines[62], High Frequency Trade markets (HFT)[63], Intelligent Traffic Systems (ITS)[64], [65], Medical visual analytics[1] and social networks[66], [67].

The recent advancements and optimizations for map reduce clusters have dramatically reshaped the options and paved the way for distributed machine learning and advanced analytics. We have walked through the time-line of various optimization techniques undertaken through the literature and provided a comparative review with respect to response-time, throughput, fault-tolerance, scalability and performance.

While different optimizations have dramatically bridged the gap between the old traditional map reduce batch-computation approach and real-time stream computations on big data, each approach has its own advantages and disadvantages. Connecting the dots between the limitations of each approach could potentially yield to a better more-mature platform for real-time analytics. We have observed that most of the benchmarks are not comprehensive and fair enough to compare different approaches. Therefore, in a future project we plan to benchmark and audit the performance of some of the techniques discussed through a standardized benchmark and a fair data set. Similarly, parallel machine learning algorithms on online streaming data are rarely present. Consequently, we plan to investigate some analytic algorithms and tune them for the future: streaming live data. Our next steps might include abstracting the computational flow of data streams using declarative programming techniques such as Constraint Handling Rules (CHR).

## References


[1] F. Wang, R. Lee, Q. Liu, A. Aji, X. Zhang, and J. Saltz, "Hadoop-gis: A high performance query system for analytical medical imaging with mapreduce," tech. rep., Technical report, Emory University, 2011.

[2] Intel, "Bigdata101: Unstructured data analytics." [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/big-data-101-brief.pdf, June 2012.

[3] M. Hogan, F. Liu, A. Sokol, and J. Tong, "Nist cloud computing standards roadmap," *NIST Special Publication*, p. 35, 2011.

[4] F. Michael, C. Mike, E. Christopher, and S. Josh, "Massive data analytics and the cloud: A revolution in intelligence analysis." [Online]. Available: http://www.boozallen.com/media/file/MassiveData.pdf, 2011.

[5] P. Russom, "big data analytics," *TDWI Best Practices Report, 4 th Quarter 2011*, 2011.

[6] S. Philpott and T. B. CoE, "Advanced analytics," 2010.

[7] N. R., "Get analytics right from the start." [Online]. Available: http://www.dnm.ie/documents/whitepapers/Analyticsfromthestart.pdf, February 2010.

[8] J. T., "Predictive analytics in the cloud." [Online]. Available: http://smartdatacollective.com/sites/default/docs/SDC-3twn0H3IkKM5peAzhvQ5/Predictive%20Analytics%20Cloud%20Position%20Paper.pdf, 2011.

[9] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[10] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, pp. 29–43, ACM, 2003.

[11] D. Borthakur, J. Gray, J. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, *et al.*, "Apache hadoop goes realtime at facebook," in *Proceedings of the 2011 international conference on Management of data*, pp. 1071–1080, ACM, 2011.

[12] A. Middleton, "Hpcc systems: Introduction to hpcc (high-performance computing cluster)," 2011.

[13] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website*, vol. 11, p. 21, 2007.

[14] G. ZHANG, C. LI, Y. ZHANG, C. XING, and J. YANG, "Mapreduce++: Efficient processing of mapreduce jobs in the cloud," *Journal of Computational Information Systems*, vol. 8, no. 14, pp. 5757–5764, 2012.

[15] J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguadé, M. Steinder, and I. Whalley, "Performance management of mapreduce applications," 2009.

[16] K. Kambatla, A. Pathak, and H. Pucha, "Towards optimizing hadoop provisioning in the cloud," in *Proc. of the First Workshop on Hot Topics in Cloud Computing*, 2009.

[17] H. Herodotou, F. Dong, and S. Babu, "Mapreduce programming and cost-based optimization? crossing this chasm with starfish," *Proceedings of the VLDB Endowment*, vol. 4, no. 12, 2011.

[18] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. Cetin, and S. Babu, "Starfish: A self-tuning system for big data analytics," in *Proc. of the Fifth CIDR Conf*, 2011.

[19] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and E. Ayguadé, "Resource-aware adaptive scheduling for mapreduce clusters," *Middleware 2011*, pp. 187–207, 2011.

[20] T. Sandholm and K. Lai, "Mapreduce optimization using regulated dynamic prioritization," in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, pp. 299–310, ACM, 2009.

[21] L. Phan, Z. Zhang, Q. Zheng, B. Loo, and I. Lee, "An empirical analysis of scheduling techniques for real-time cloud-based data processing," in *IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 2011*, pp. 1–8, IEEE, 2011.

[22] D. Jiang, B. Ooi, L. Shi, and S. Wu, "The performance of mapreduce: An in-depth study," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 472–483, 2010.

[23] C. Gkantsidis, D. Vytiniotis, O. Hodson, D. Narayanan, and A. Rowstron, "Automatic io filtering for optimizing cloud analytics microsoft technical report msr-tr-2012-3,"

[24] A. Thusoo, J. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive-a petabyte scale data warehouse using hadoop," in *IEEE 26th International Conference on Data Engineering (ICDE), 2010*, pp. 996–1005, IEEE, 2010.

[25] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1099–1110, ACM, 2008.

[26] A. Khetrapal and V. Ganesh, "Hbase and hypertable for large scale distributed storage systems," *Dept. of Computer Science, Purdue University*, 2006.

[27] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[28] A. Lakshman and P. Malik, "Cassandra: A structured storage system on a p2p network," in *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pp. 47–47, ACM, 2009.

[29] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Applied statistics*, pp. 100–108, 1979.

[30] D. E. Goldberg, "Genetic algorithms in search, optimization, and machine learning," 1989.

[31] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. Bae, J. Qiu, and G. Fox, "Twister: a runtime for iterative mapreduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 810–818, ACM, 2010.

[32] Y. Bu, B. Howe, M. Balazinska, and M. Ernst, "Haloop: Efficient iterative data processing on large clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 285–296, 2010.

[33] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears, "Online aggregation and continuous query support in mapreduce," in *ACM SIGMOD*, pp. 1115–1118, 2010.

[34] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pp. 21–21, 2010.

[35] C. Olston, G. Chiou, L. Chitnis, F. Liu, Y. Han, M. Larsson, A. Neumann, V. B. Rao, V. Sankarasubramanian, S. Seth, *et al.*, "Nova: continuous pig/hadoop workflows," in *Proceedings of the 2011 international conference on Management of data*, pp. 1081–1090, ACM, 2011.

[36] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pp. 10–10, USENIX Association, 2010.

[37] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2011.

[38] J. Boulon, A. Konwinski, R. Qi, A. Rabkin, E. Yang, and M. Yang, "Chukwa, a large-scale monitoring system," in *Proceedings of CCA*, vol. 8, 2008.

[39] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *IEEE International Conference on Data Mining Workshops (ICDMW), 2010*, pp. 170–177, IEEE, 2010.

[40] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik, "Scalable distributed stream processing," in *Proc. Conf. on Innovative Data Syst. Res*, 2003.

[41] M. Nathan, X. James, and J. Jason, "Storm: Distributed real-time computation system." [Online]. Available: http://storm-project.net/, 2012.

[42] "Hstreaming." [Online]. Available: http://www.hstreaming.com/, 2011.

[43] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: interactive analysis of web-scale datasets," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 330–339, 2010.

[44] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters," in *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, pp. 10–10, USENIX Association, 2012.

[45] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: A fault-tolerant model for scalable stream processing," tech. rep., UC Berkeley Technical Report UCB/EECS-2012-259, 2012.

[46] M. Sax, M. Castellanos, Q. Chen, and M. Hsu, "Aeolus: An optimizer for distributed intra-node-parallel streaming systems," in *Demo), Will be presented at Int. Conf. on Data Engineering*, 2013.

[47] C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," *Advances in neural information processing systems*, vol. 19, p. 281, 2007.

[48] S. Venkataraman, I. Roy, R. S. Schreiber, and A. AuYoung, "Presto: Complex and continuous analytics with distributed arrays," 2011.

[49] T. H. T. D. M. Zaharia, A. Bayen, and P. Abbeel, "Large-scale online expectation maximization with spark streaming,"

[50] K.-C. Lee and D. Kriegman, "Online learning of probabilistic appearance manifolds for video-based recognition and tracking," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005.*, vol. 1, pp. 852–859, IEEE, 2005.

[51] M. D. Hoffman, D. M. Blei, and F. Bach, "Online learning for latent dirichlet allocation," *Advances in Neural Information Processing Systems*, vol. 23, pp. 856–864, 2010.

[52] R. S. Sutton, S. D. Whitehead, *et al.*, "Online learning with random representations," in *Proceedings of the Tenth International Conference on Machine Learning*, pp. 314–321, Citeseer, 1993.

[53] S. Shalev-Shwartz, Y. Singer, and A. Y. Ng, "Online and batch learning of pseudo-metrics," in *Proceedings of the twenty-first international conference on Machine learning*, p. 94, ACM, 2004.

[54] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *The Journal of Machine Learning Research*, vol. 7, pp. 551–585, 2006.

[55] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online learning with kernels," *Signal Processing, IEEE Transactions on*, vol. 52, no. 8, pp. 2165–2176, 2004.

[56] S. Shalev-Shwartz, "Online learning: Theory, algorithms, and applications," 2007.

[57] A. Basak, I. Brinster, and O. J. Mengshoel, "Mapreduce for bayesian network parameter learning using the em algorithm," *Proc. of Big Learning: Algorithms, Systems and Tools*, 2012.

[58] E. B. Reed and O. J. Mengshoel, "Scaling bayesian network parameter learning with expectation maximization using mapreduce," *Proc. of Big Learning: Algorithms, Systems and Tools*, 2012.

[59] A. G. Schwing, T. Hazan, M. Pollefeys, and R. Urtasun, "Distributed structured prediction for big data,"

[60] J. Ye, J.-H. Chow, J. Chen, and Z. Zheng, "Stochastic gradient boosted distributed decision trees," in *Proceedings of the 18th ACM conference on Information and knowledge management*, pp. 2061–2064, ACM, 2009.

[61] S. Daruru, N. M. Marin, M. Walker, and J. Ghosh, "Pervasive parallelism in data mining: dataflow solution to co-clustering large and sparse netflix data," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1115–1124, ACM, 2009.

[62] D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications," in *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, 2010.

[63] S. Arnuk and J. Saluzzi, "Latency arbitrage: The real power behind predatory high frequency trading," *Themis Trading white paper*, 2009.

[64] L. D. Baskar, B. De Schutter, and H. Hellendoorn, "Model-based predictive traffic control for intelligent vehicles: Dynamic speed limits and dynamic lane allocation," in *Intelligent Vehicles Symposium, 2008 IEEE*, pp. 174–179, IEEE, 2008.

[65] B. L. Smith and M. J. Demetsky, "Short-term traffic flow prediction models-a comparison of neural network and nonparametric regression approaches," in *IEEE International Conference on Systems, Man, and Cybernetics, 1994.'Humans, Information and Technology'., 1994*, vol. 2, pp. 1706–1709, IEEE, 1994.

[66] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM SIGMOD Record*, vol. 34, no. 4, pp. 42–47, 2005.

[67] B. Aleman-Meza, M. Nagarajan, L. Ding, A. Sheth, I. B. Arpinar, A. Joshi, and T. Finin, "Scalable semantic analytics on social networks for addressing the problem of conflict of interest detection," *ACM Transactions on the Web (TWEB)*, vol. 2, no. 1, p. 7, 2008.