Survey of Real-time Processing Systems for Big Data

Xiufeng Liu
University of Waterloo
Canada
xiufeng.liu@uwaterloo.ca

Nadeem Iftikhar
University College of Northern
Denmark
naif@ucn.dk

Xike Xie
Aalborg University
Denmark
xkxie@cs.aau.dk

ABSTRACT

In recent years, real-time processing and analytics systems for big data–in the context of Business Intelligence (BI)–have received a growing attention. The traditional BI platforms that perform regular updates on daily, weekly or monthly basis are no longer adequate to satisfy the fast-changing business environments. However, due to the nature of big data, it has become a challenge to achieve the real-time capability using the traditional technologies. The recent distributed computing technology, MapReduce, provides off-the-shelf high scalability that can significantly shorten the processing time for big data; Its open-source implementation such as Hadoop has become the de-facto standard for processing big data, however, Hadoop has the limitation of supporting real-time updates. The improvements in Hadoop for the real-time capability, and the other alternative real-time frameworks have been emerging in recent years. This paper presents a survey of the open source technologies that support big data processing in a real-time/near real-time fashion, including their system architectures and platforms.

Categories and Subject Descriptors

H.2.7 [**Database Administration**]: Data warehouse and repository; H.2.m [**Miscellaneous**]

Keywords

Survey, Real-time, Big data, Architectures, Systems

1. INTRODUCTION

Today's business intelligence systems increasingly demand the support for the integration and analytics of big data. More and more enterprises are facing the challenges of handling fast-growing amount of data and data diversity. The traditional BI is not sufficient in dealing with these challenges. Thanks to the cloud computing technologies such as PaaS and SaaS, many users start to move their BI solutions to the cloud platform that provides un-limited computing resources and storage capacity [20, 21, 22]. The distributed computing frameworks such as MapReduce [8] are now widely used to process big data. MapReduce makes use of hundreds and thousands of "pluggable" nodes in a cluster to process data in parallel, which greatly shortens the time between the operational events and presenting the analytics results. Since its debut in 2004, MapReduce has been attracting an ever-growing popularity due to its cost-effective, high scalability and fault-tolerance, and now it has become the de-facto technology for big data. A growing number of BI vendors have embraced MapReduce, and introduced it into their platforms, including Informatica, Pentaho, IBM Websphere and Oracle Data Warehouse, and many others. In the open source world, many MapReduce-based systems have been developed, including Pig, Hive, Cascade, etc., and the auxiliary software such as flume (the tool for streaming data integration), Ganglia (the tool for administrating a cluster), and Sqoop (an integration tool with Hadoop cluster), etc. In addition, there are some other non-MapReduce-based that co-exist with MapReduce systems to deal with big data, such as the NoSQL databases, MongoDB, ElephantDB, Cassandra, Hbase, etc. These two classes of the software products have formed the software ecosystem for big data. It is not un-common to see an enterprise software stack containing up to tens of products. The diverse stacked software poses a high learning curve and challenge in management, which requires sorting out for a better use purpose.

ref According to the recent big data survey over 274 business and IT decision-makers in Europe [6], there is a clear trend towards making data available for analysis in (near) real-time, and over 70% of responders indicate the need for real-time processing. However, most of the existing big data technologies are designed to achieve high throughput, but not low latency, probably due to the nature of big data, i.e., high volume, high diversity and high velocity. The survey also shows that only 9% of the companies have made progress with faster data availability, due to the complexity and technical challenges of real-time processing. Most analytic updates and re-scores have to be completed by long-run batch jobs, which significantly delays presenting the analytic results. Therefore, the whole life-cycle of data analytics systems (e.g., business intelligence systems) requires innovative techniques and methodologies that are capable of providing real-time or near real-time results. That is, from the capturing of real-time business data to transformation and delivery of actionable information, all these stages in the life-cycle of data analytics require value-added real-time functionalities, including real-time data feeding from operational sources, ETL optimization, and generating real-time analytical models and dashboards.

Since the advances toward real time are affecting many enterprise applications, an increasing number of systems have emerged to support real-time data integration and analytics in the past few years. Some of the technologies tweaked the existing cloud-based products to lower the latency when processing large-scale data sets; while some others came along with the other products to consti-

IDEAS '14 July 07 - 09 2014, Porto, Portugal

http://dx.doi.org/10.1145/2628194.2628251.

tute a real-time processing system. This has brought the new dawn for enterprises to tackle with the real-time challenge. We believe it is necessary to make a survey of the existing real-time processing systems. In this paper, we make the following contributions: First, the paper reviews the MapReduce Hadoop implementation. It discusses the reasons why Hadoop is not suitable for real-time processing. Then, the paper surveys the real-time architectures and open source technologies for big data with regards to the following two categories: data integration and data analytics. In the end, the paper compares the surveyed technologies in terms of architecture, usability, and failure recoverability, etc., and also discusses the open issues and the trend of real-time processing systems.

The rest of the paper is structured as follows. Section 2 reviews Hadoop framework. Section 3 reviews real-time data integration and analytics technologies. Section 4 surveys the current widely-used real-time architectures, and the real-time systems. Section 7 discusses open issues and challenges of real-time capability, and compares the investigated systems. Section 8 concludes the paper.

2. BIG DATA ENGINE – HADOOP

Hadoop [12] is an open-source implementation of MapReduce by Doug Cutting and Mike Cafarella. Hadoop can scale out the data processing on many commodity machines. Hadoop is now commonly considered as the engine of big data platforms.

Hadoop consists of two core components: Hadoop Distributed File System (HDFS); and the MapReduce programming and job management framework. HDFS and MapReduce both employ master-slave architecture. A Hadoop program (or client) submits jobs to the MapReduce framework through the jobtracker running on the master server. The jobtracker assigns the tasks to tasktrackers running on many slave nodes. The tasktrackers send heartbeats regularly to the jobtracker to update the status, such as alive, idle or busy, etc. If a task fails or timeouts, or a node is dead, the jobtracker can re-schedule the tasks to run on available nodes automatically. HDFS component consists of one namenode and multiple datanodes. The namenode keeps the metadata about the data on each datanode. When a client application reads or writes data into HDFS, it first has to communicate with the namenode to get the locations of data block to be read from, or written to. The metadata is read into main memory when Hadoop starts, and is maintained dynamically. A datanode updates namenode the metadata of its local data blocks through heart beats. Hadoop also has a secondary namenode, but it is not the standby node for shooting a single point of failure for the namenode (the name is somewhat misleading). The secondary namenode is used to store the latest checkpoints of HDFS states.

The Limitations. The Hadoop MapReduce framework is designed with the scalability and fault-tolerance as the goal, and is not optimized for I/O efficiency [18]. Map and Reduce both are the block operations, in which data transition cannot proceed to the next stage until the tasks of the current stage have finished. The output of mappers has to be first written into HDFS before shuffled to the reducers. The shuffling will not start until all the map tasks have finished, due to the sort-merge based grouping for the intermediate results. The block-level restart, a one-to-one shuffling strategy, and the runtime scheduling lower the performance of each node. The MapReduce framework lacks the execution plans like DBMS, and does not optimize data transferring across different nodes. Therefore, Hadoop has a latency problem due to its inherent nature. Thus, it is more suitable for batch jobs than real-time processing.

3. DATA INTEGRATION AND ANALYTICS

The data process, e.g., in business intelligence systems, typically consists of two stages, data integration and data analytics.

Data integration. Data integration typically is referred to the process of data extraction-transformation-load (ETL). Traditionally this process runs at a regular time interval, such as daily, weekly or monthly. In the real-time scenario, however, large amounts of small-sized data are processed into the data warehouse, possibly in high velocity and in many parallel streams. The “big" of this scenario is because the real-time data is aggregated, and fed to data warehouse continuously. A typical example is the algorithmic trading in markets, also called automated trading, which leverages real-time technologies coupled with proprietary algorithms to execute trading orders automatically. The algorithmic trading of “high-frequency trading" can make the decision to initiate orders based on the received real-time information. Another example is electronic commercial websites such as eBay, Amazon, and Taobao, which work with a large retailer. The website traffic is made of a lot of web-click stream events, and innumerate online transactions. The click streams feed a large amount of data into the underlying data warehouse in a real-time fashion, augmented by many other data streams, such as social, inventory, CRM, call center, in-store transactions, etc. Therefore, these scenarios and the similar, require the innovative technologies, which support processing the scalable data continuously and timely. While, the traditional technologies of running data process at regular intervals is typically not applicable.

Real-time analytics. Data analytics goes after the integration process, which analyzes the data residing in a data store, or in streaming. Data analytics is to get the insight/value of data with the help of analytics tools. When the data size grows to web scale, using a traditional analytic tool becomes challenging, e.g., not able to return the analytics results within a time limit, thus loses the value. Therefore, an increasing number of analytic tools emerged in the past two years to cater for the real-time requirement. These tools address the need to have “as fast as disk" processing speeds, or “near real time". The tools can capture, query and present analytic results within seconds to minutes. For example, some analytics engines start to support applying the analytic techniques that are used for static (stored) data to the moving streams. End-users and business analysts can access/-query both streaming and historical data in a real-time manner. Another evidence is the rise of NoSQL databases, which are intensively used in performance-critical scenarios. NoSQL databases generally are key-value store, and exhibit the characteristics of schema free, easy replication support, eventually consistent, supporting a huge amount of data, and good efficiency.

4. REAL-TIME SYSTEM ARCHITECTURE

We are witnessing the shift in data processing from the batch-based to the real-time based. A lot of technologies can be used to create a real-time processing system, however, it is complicated and daunting to choose the right tools, incorporate, and orchestrate them. We are now introducing a generic solution to this problem, called *Lambda Architecture* proposed by Marz [23]. This architecture defines the most general system of running arbitrary functions on arbitrary data using the following equation “*query = function(all data)*". The premise behind this architecture is that one can run ad-hoc queries against all the data to get results, however, it is very expensive to do so in terms of resource. So the idea is to pre-compute the results as a set of views, and then query the views.

Figure 1 describes the lambda architecture. The architecture consists of three layers. First, the batch layer computes views on the collected data, and repeats the process when it is done to infinity. Its output is always outdated by the time it is available for new data has been received in the meantime. Second, a parallel speed processing layer closes this gap by constantly processing the most recent data

in near real-time fashion. Any query against the data is answered through the serving layer, i.e., by querying both the speed and the batch layers' serving stores in the serving layer, and the results are merged to answer user queries.

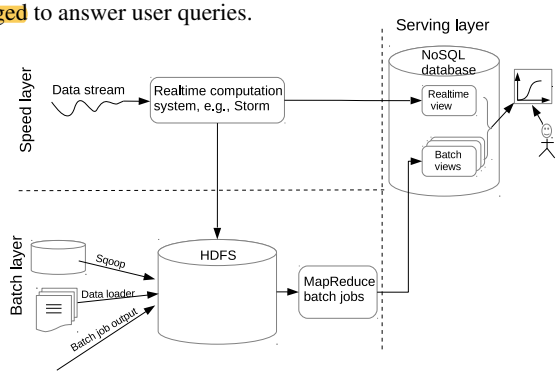

Figure 1: Three-layered lambda architecture

The batch and speed layers both are forgiving and can recover from errors by being recomputed, rolled back (batch layer), or simply flushed (speed layer). A core concept of the lambda architecture is that the (historical) input data is stored unprocessed. This concept provides the following two advantages. Future algorithms, analytics, or business cases can retroactively be applied to all the data by simply adding another view on the whole data. Human errors like bugs can be corrected by re-computing the whole output after a bug fix. Many systems are built initially to store only processed or aggregated data. In these systems, errors are often irreversible. The lambda architecture can avoid this by storing the raw data as the starting point for the batch processing.

However, the lambda architecture itself is only a paradigm. The technologies with which the different layers are implemented are independent from the general idea. The speed layer deals only with new data and compensates for the high latency updates of the batch layer. It can typically leverage stream processing systems, such as Storm, S4, and Spark, etc. The batch serving layer can be very simple in contrast. It needs to be horizontally scalable and supports random reads, but is only updated by batch processes. The technologies like Hadoop with Cascading, Scalding, Python streaming, Pig, and Hive, are suitable for the batch layer. The serving layer requires a system that has the ability to perform fast random reads and writes. The data in the serving layer are replaced by the fresh real-time data, and the views computed by the batch layer. Therefore, the size of data is relatively small, and the system in serving layer does not need complex locking mechanisms since the views are replaced in one operation. In-memory data stores like Redis or Memchache, are sufficient for this layer. If users require high availability, low latency and storing large volume of data in the serving store, the systems like HBase, Cassandra, ElephantDB, MongoDB, or DynamoDB can be the potential option.

Framework that can be used in serving layer

5. REAL-TIME PLATFORMS

We now introduce the platforms that are widely used for getting real-time availability for big data.

Hadoop Online. Due to the limitations of Hadoop for the real-time processing, the blocking behavior of the mappers and the reducers needs to be eliminated to support continuous processing. Condie et al. implement Hadoop Online [7], which can pipeline the intermediate data between Map and Reduce operators. Hadoop Online supports both the pipeline within one MapReduce job and between consecutive jobs. Different to Hadoop where a reducer reads the data from mappers in a pull fashion, a mapper in Hadoop Online transmits data to reducers in a push fashion. That is, whenever a mapper has finished processing a key/value pair, it directly sends the data to the reducers through socket connections. Therefore, the reducers in Hadoop Online do not have to wait until the last map task has finished. Hadoop Online also supports transferring data from the reducers directly to the mappers of the next job. This is different to Hadoop, which requires a job first to write the data into HDFS, then the subsequent job reads the intermediate data from HDFS. Thus, the pipelining in Hadoop Online greatly lowers the latency of transferring data between a job to another.

Therefore, Hadoop Online can be used for complex continuous queries that incur several MapReduce jobs. The prototype of Hadoop Online is based on Hadoop 0.19.2, and this prototype supports all the features and configuration parameters supported by Hadoop 0.19.2.

Storm. We now introduce the data streaming framework, Storm [30]. Storm is an open source low latency data stream processing system, which integrates with other queuing and bandwidth systems. Storm consists of several moving parts, including the coordinator (ZooKeeper), state manager (Nimbus), and processing nodes (Supervisor). Storm implements the data flow model in which data flows continuously through a network of transformation entities. The abstraction for a data flow is called *stream*, which is an unbounded sequence of tuples. A tuple is the data structure for representing the standard data types (such as ints, floats, and byte arrays) or user-defined types with some additional serialization code. A stream is used to build data processing topologies between data sources and sinks. Streams originate from *spouts*, which flow data from external sources into the Storm topology. The sinks (or entities that provide transformations) are called *bolts*, each of which implements a single transformation on a stream. Bolts can implement traditional things like filtering, aggregation, and output writer; and more complex actions like MapReduce. A typical Storm topology implements multiple transformations, and therefore requires multiple bolts with independent tuple streams.

Storm runs in-memory, and is therefore set to process large volumes of data at in-memory speed. However, a typical storm architecture needs to interface with a data source for its input and a data store for its output. Quite often these interfaces rely on file-based message queues for streaming, and NoSQL for data storage, which are at least an order of magnitude slower than Storm itself. In this way, the memory-based middleware, e.g., Trident, can be used to alleviate the compromise at the two integration points.

Today, many enterprises use Storm combined with HBase as their real-time architecture for processing streaming data. In this architecture, Storm is used to integrate streaming data continuously, while HBase is used as the data store for saving data, and for queries. Since HBase is the open source project modeled after Google's big data, it optimizes both read and write. HBase uses the built-in data buffer, *MemStore*, to accumulate data in memory before a permanent write. The data buffer is connected to a storm bolt of the data stream. For queries, HBase makes use of the LRU cache, *BlockCache*, to optimize read performance.

Flume. Flume [3] is a distributed, reliable, and available system for efficiently collecting, aggregating, and moving large amounts of event data. The topology of Flume is made of multiple agents, each of which runs in a separate Java Virtual Machine (JVM). An agent consists of three pluggable components, named *source*, *sink* and *channel*. Source collects incoming data as events, sink writes events out, and channels connect the source and sink. The channel on each agent functions as the data buffer that stores the events in case of the downstream failure or shutdown. Flume has file-based and memory-based channels. The file-based is a durable channel that can persist events to disk. Therefore, if a JVM is killed, or the operating system crashes or reboots, the events that were not successfully transferred

to the next agent are still able to be recovered. The memory-based channel is a volatile channel that any events stored in the memory channel are lost if the above situation occurs. But, the benefit of using memory-based channels is to get a lower latency since it does not have to write the events to disk in a transaction. Flume also supports batch events. The batch size is the maximum number of the events that a sink or a client attempts to take from a channel in a single transaction. With a small batch size, the throughput decreases, but the risk of event duplication is reduced if a failure were to occur. With a large batch size, users can get a much higher throughput, but increase the latency, and in the case of a transaction failure, the number of possible duplicates increases. Therefore, users can tune the batch size by trading off the throughput vs. latency and duplication under failure.

The original purpose of Flume is used to log data aggregation. But, since data sources are customizable, Flume can be used to transport massive quantities of event data, such as network traffic data, social-media-generated data, and email messages. The sink typically writes events to HDFS. But, since Flume agent has the pluggable capability for the output, users can implement the agent interface to support any types of the databases for writing data.

Spark and Spark Streaming. Spark [28] is a cluster computing system originally developed by UC Berkeley AMPLab. Now it is an umbrellaed project of Apache foundation. The aim of Spark is to make data analytics program run faster by offering a general execution model that optimizes arbitrary operator graphs, and supports in-memory computing. This execution model is called Resilient Distributed Dataset (RDD), which is a distributed memory abstraction of data. Spark performs in-memory computations on large clusters in a fault-tolerant manner through RDDs [32]. Spark can work on the RDDs for multiple iterations which are required by many machine learning algorithms. A RDD resides in main memory, but can be persisted to disk as requested. If a partition of RDD is lost, it can be re-built. Spark also supports shared variable, broadcast variable and accumulator variable. A shared variable is typically used in the situation where a global value is needed, such as lookup tables, and number counter.

Spark Streaming [28] is extended from Spark by adding the ability to perform online processing through a similar functional interface to Spark, such as map, filter, reduce, etc [33]. Spark Streaming runs streaming computations as a series of short batch jobs on RDDs, and it can automatically parallel the jobs across the nodes in a cluster. It supports fault recovery for a wide array of operators.

Kafka. Kafka [5, 11, 17] is a real-time message publish/subscribe system developed by LinkedIn. Kafka maintains the feeds of messages in topic categories, each category has several partitions, and each partition contains an ordered, immutable sequence of messages that is continually appended. Each message is assigned a unique sequential id for identifying the message in a partition. In a Kafka cluster, a partition is distributed over multiple nodes for fault-tolerance. A Kafka cluster retains the published messages for a configurable period of time. When the time is due, the messages are discarded no matter they have been consumed or not. In message subscription, each consumer is labeled with the group name of the consumer belonging to. A partition of a topic can only be delivered to one consumer in a consumer group, which ensures the correct message order.

The original use of Kafka is to rebuild a user activity tracking pipeline as a set of real-time publish/subscribe feeds [5]. The site activities, such as browse pages, search, and the other actions users may take, are published to a Kafka cluster as topics, one topic per activity type. The topics are available to subscription for a range of use cases, including real-time processing, real-time monitoring, and loading data into Hadoop or data warehouses. The data pipelines in Kafka are monitored, e.g., monitoring the statistics information of the aggregations from distributed applications for producing centralized feeds of operational data. Therefore, Kafka is well suited for the situations where users need to process real-time data, and analyze them. Currently, Kafka at LinkedIn supports dozens of subscribing systems, and delivers more than 55 billion messages to consumers per day [11].

Scribe. Scribe [27] is the system for aggregating streaming log data. It is designed to scale to a very large number of nodes and be robust to network and node failures. Scribe running on each node is configured to aggregate messages, and to send messages to a central Scribe server (or multiple Scribe servers in a large cluster). If the central Scribe server is not available, Scribe will write the messages to the local disk and sends them when the central server recovers. The central Scribe server(s) write the messages to the files as the final destination, typically on a network file system or a distributed file system, or send them to another layer of Scribe servers.

Scribe is unique in that clients can log the entries containing both category and message. The category is a high level description of the destination that messages are sent to, and can have a specific configuration in a Scribe server. This allows data stores to be moved to other places, by changing Scribe configuration instead of client code. The server also allows to use the configuration set based on category prefix, and use the default configuration with category names in the file path. Flexibility and extensibility are made through the “store" abstraction. Stores are loaded dynamically based on a configuration file, and can be changed at run-time without stopping the server. Stores have a class-like hierarchy, and a store can contain the other stores. This allows users to combine and chain the features in a different order (simply by changing the configuration). Scribe is implemented as a thrift service using a non-blocking C++ server. The use of Scribe at Facebook runs on thousands of machines, and reliably delivers tens of billions of messages per day [27].

S4. S4 is a general-purpose, near real-time, distributed, decentralized, scalable, event-driven, modular platform for processing continuous unbounded streaming data [26, 25]. S4 has a decentralized and symmetric architecture which all the nodes in a cluster are identical, different to the classic master-nodes architecture. S4 employs ZooKeeper as the communication layer to coordinate the nodes within the cluster. But, this is transparent to users since S4 can automatically handle communication, scheduling and distribution across all the nodes. The design of S4 is derived from the combination of MapReduce and Actors model [13]. The computation on S4 is performed by the so-called *Processing Elements (PEs)*. S4 transmits messages between the PEs in the form of data events. The state of each PE is inaccessible to the other PEs, and the event emission and consumption are the only mode of interaction between PEs [25]. An S4 cluster contains many PEs to process events. Since data is streamed between PEs, no on-disk checkpoint is required. Thus, only partial fault tolerance is supported in S4. For example, if a processing node fails, its processes are automatically moved to a standby server, but the states of these processes are lost, and cannot be recovered. This shares a great similarity with Storm (but Storm uses master-slaves architecture instead).

HStreaming. HStreaming is an analytics platform built on top of Hadoop and MapReduce [15]. The architecture of HStreaming consists of two components: data acquisition and data analytics. The data acquisition component is able to collect data in near real-time, and has ETL capabilities, while the analytics component allows to analyze unstructured and structured data on HDFS in a real-time fashion. HStreaming also provides the connectors for connecting both SQL and NoSQL data stores, which make it possible to ana-

lyze the data from different databases. HStreaming provides both enterprise and community editions.

All-RiTE. All-RiTE [19] is an ETL middleware system (extended from [31]), which enables efficient processing live data warehouse (DW) data. The *live DW data*, such as *accumulating facts* and *early-arriving facts*, will eventually be updated to the data warehouse, but also queried in an online fashion. It is typically not efficient to deal with live data using traditional technologies, such as using SQL INSERTs followed by the possible updates and deletions, when the data has been loaded into the DW. All-RiTE solves this problem by making use of an intermediate data store, called *catalyst*. The catalyst locates between data producers and the DW, and accumulates live data in the in-memory store. Therefore, if there necessitates to update or delete the rows in the catalyst, the updates and deletions are done on-the-fly when the data is queried or materialized to the DW. All-RiTE exploits a number of user-defined *flush policies* to decide *when* to move data from source systems towards the DW. Through the flush policies, the data could be queried in a near real-time or right-time fashion (right-time means only the data satisfying the specified time accuracy is read). All-RiTE can run in stand-alone mode or be integrated with other ETL tools to process live DW data.

Impala. Impala is an open source real-time analytics system developed by Cloudera [16]. This system is inspired by Google's Dremel [24], which is a scalable, interactive ad-hoc query system for data analytics. Impala provides an SQL-like engine to execute queries against the data in HDFS and HBase, which is somewhat similar to Hive. But, unlike Hive which relies MapReduce to execute queries, Impala does not use Hadoop at all, which makes a significant performance improvement for queries. In an Impala cluster, Impala daemons run on all the nodes, which cache some of the data read from HDFS, and process the in-memory data. Therefore, the results can be returned very quickly. Impala uses the metastore of Hive to save the metadata, and supports the subset of Hive SQL, ODBC driver and friendly user-interfacing (through Hue Beeswax). Therefore, to Hive users, Impala provides a familiar and unified platform for both batch-oriented and real-time queries. The first beta version supports text files and SequenceFiles, and additional formats including Avro, RCFile, LZO text files, and new Parquet columnar format. However, Impala does not provide fault-tolerance compared to Hive due to its use of the in-memory based operations.

6. MESSAGING TECHNOLOGIES

To complement this survey, we make a discussion about the messaging technologies, which are widely used in real-time processing systems. Currently, a lot of messaging technologies are available, including the proprietary and the open source. The most popular open source ones include ActiveMQ, RabbitMQ, ZeroMQ and HornetQ. Typically, each of these messaging technologies has a built-in message queue to support asynchronous communication. Therefore, message senders and receivers do not have to interact with each other directly, but through the messages placed into a queue. A message sender does not have to wait for the message recipient to finish the computing and returning the results, instead, the results are returned to the message sender later by a callback function. The asynchronous messaging technologies are used extensively in responsive systems, such as in interactive data analysis and HTML5 technology. Facebook is a good example, where a single page is divided into multiple portlets, each of which can retrieve the information from the underlying database systems separately, and shows the results asynchronously. Storm uses ZeroMQ as the default messaging technology to inter-worker communication. Some benchmark data are available for the listed messaging systems such as [1]. The benchmark includes the ranking of the popularity, the best use cases, performance and the support of persistence.

7. DISCUSSION

MapReduce paradigm and its implementation, Hadoop, are widely used in data-intensive area, and receive ever-growing attentions. They are the milestone to the era of big data. Since MapReduce was originally invented to process large-scale data in batch jobs in a shared-nothing cluster environment, the implementation is very different to the traditional processing systems such as DBMSs. The scalability, efficiency and fault-tolerance are the main focuses for MapReduce implementations, instead of real-time capability. In the past few years, many technologies were proposed to optimize MapReduce-based systems, including Hadoop Online [7], Hadoop indexes (eg., Hadoop++ [9]), column-oriented storage (e.g., RCFile [14] and Dremel [24]), Co-location (e.g., CoHadoop [10]) data compression, etc. These technologies narrow the "gap" between batch and real-time data processing. However, we believe that the decision of selecting the best data processing system, depending on the types and the sources of data, and the processing time needed, and the ability to take immediate actions. MapReduce paradigm is best suited for batch jobs. For example, Abadi et al. [2] have done the benchmark study of comparing MapReduce paradigm and parallel DBMSs, and the results show that DBMSs substantially faster than MapReduce. Stonebraker also points out that MapReduce style systems excel at complex analytics and ETL tasks [29].

In view of the deficiency regarding real-time capability in Hadoop 1.x, the next generation Hadoop, YARN [4], is implemented as the open framework that can integrate with different third-party real-time processing systems, including Storm, Flume, Spark, etc, which is a promising direction to achieve real-time capability. By the help of the third-party components, YARN compensates for the deficiency of real-time capability in Hadoop 1.x. However, at the time of writing this paper, we still have not found that any real-world cases of using YARN for doing real-time analytics. To the best of our knowledge, most of the current real-time enterprise systems are using the three-layer lambda architecture or its variants as discussed in Section 4. The main reason that makes the lambda architecture popular is its highly flexibility, in which users can combine different technologies catering for their business requirements, e.g., in the context of legacy systems, the lambda architecture can show its advantage over others.

The requirement of real-time processing of high-volume data streams brings a revolution to the traditional data processing infrastructures. The data streams include ticks of market data, network monitoring, fraud detection, and command and control in military environments, etc. The real-time processing and analytics allow a business user to take immediate action in response to the events of streams that are significant. It is vital to optimize the data stream processing technologies, such as to minimize disk I/O and communication. Now many real-time processing systems seek ways to enable in-memory processing to avoid reading to/from disk whenever possible, as well as integration with distributed computing technologies. This improves the scalability while maintaining a low latency. In Table 1, we compare the real-time processing systems we investigated. This table lists the types of architectures, their supports of the catalog (metadata), data integration and analytics, recoverability, in-memory capability, and license types. As shown in the table, Storm, Flume, S4 and Impala use peer architecture in which all the nodes are identical, in contrast to master-slave architecture. Except Impala, which uses Hive MetaStore to save its metadata, all the others do not have a separate catalog store. Spark Streaming and HStreaming both support integration and analytics queries. Impala supports analytics queries, and the others only support the integra-

Table 1: Comparative table of real-time processing systems

Name of system	Architecture	Catalog	Integration/ Query	In-memory	Recovery	License
Hadoop Online	Master/slaves	No	Integration	Yes	No	Apache License 2.0
Storm	Peer	No	Integration	Yes	Manual	Eclipse Public License
Flume	Peer	No	Integration	Yes/ backed by files	Manual	Apache License 2.0
Spark Streaming	Master/slaves	No	Both	Yes	Parallel	Apache License 2.0
Kafka	publish/subscribe	No	Integration	Yes/ backed by files	Yes	Apache License 2.0
Scribe	Master/ slaves	No	Integration	Yes/ backed by files	Yes	MIT License
S4	Peer	No	Integration	Yes	Manual	Apache License 2.0
HStreaming	Master/slaves	No	Both	Semi-in-memory	Yes	Community
All-RiTE	Peer	No	Both	In-memory	Yes	GPL 2.0
Impala	Peer	Yes	Analytics query	Yes	Yes	Apache License 2.0

tion of streaming data. The table shows the evidence that almost all the systems incline to using in-memory technologies for the real-time processing, and some of them are backed by files for fault tolerance, e.g., Flume, Kafka, Scribe, and HStreaming. For the license, except for HStreaming which is a proprietary product(the community version is available), all the others have open source licenses.

During our investigation, we found that an open issue is the lack of a standard benchmark or a set of typical workloads for comparing different real-time processing systems. Only some fragmented benchmark data is available for some of the systems, e.g., [17, 34]. We believe that it would be interesting to develop a standard benchmark, and do the benchmark study to compare all of the current available real-time processing systems.

8. CONCLUSION

With the advent of the big data era, dealing with large amounts of data is challenging. MapReduce framework, such as Hadoop, provides a distributed computing platform for managing, storing and analyzing big data, but it acts as a batch processing system with a high throughput and with the drawback of high latency. Hadoop does not effectively accommodate the needs of real-time processing capability. In order to overcome this limitation, some real-time processing platforms appeared recently. The emerging systems have shown the advantage in dealing with uninterrupted data streams, and provide real-time data analytics. In this paper, we first studied the shortcomings of Hadoop in regards to the real-time data processing, then presented one of the most popular real-time architectures, the *three-layer lambda architecture*. Furthermore, We have made the survey of *open-source* real-time processing systems, including Hadoop Online, S4, Storm, Flume, Spark streaming, Kafka, Scribe, S4, HStreaming, Impala, and the relevant messaging technologies. We have compared the real-time systems from the perspectives of architectures, use cases (i.e., integration or query analytics), recoverability from failures, and license types. We also found that despite the diversity, the real-time systems share a great similarity of which heavily use main memory and distributed computing technologies. It is of great value to come up with an efficient framework for gathering, processing and analyzing big data in a near real-time/real-time fashion. Investigations into the current popular platforms and techniques provide a good reference for the users who are interested in implementing real-time systems for big data. The results also show that there is a need for benchmarking the real-time processing systems.

9. REFERENCES

[1] A Quick Message Queue Benchmark: ActiveMQ, RabbitMQ, HornetQ, QPID, Apollo, URL: atx-aeon.com/wp/2013/04/10/a-quick-message-queue-benchmark-activemq-rabbitmq-hornetq-qpid-apollo as of 2014/5/1.

[2] D. J. Abadi et al. Column-stores vs. row-stores: how different are they really? In Proc. of the SIGMOD, pp. 967–980, 2008.

[3] Apache Flume. URL: flume.apache.org as of 2014/5/1.

[4] Apache Hadoop NextGen MapReduce (YARN). URL: hadoop.apache.org as of 2014/5/1.

[5] Apache Kafka – A high-throughput distributed messaging system. URL: kafka.apache.org as of 2014/5/1.

[6] Big Data Survey Europe. URL: www.pmone.com/fileadmin/user_upload/doc/study/BARC_BIG_DATA_SURVEY_EN_final.pdf as of 2014/5/1.

[7] T. Condie et al. MapReduce Online. NSDI 10(4):20, 2010.

[8] J. Dean, and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *CACM*, 1(51):107–113, 2008.

[9] J. Dittrich et al. Hadoop++: making a yellow elephant run like a cheetah. *PVLDB*, 3(1):518–529, 2010.

[10] R. Gemulla et al. CoHadoop: flexible data placement and its exploitation in Hadoop. PVLDB, 4(9):575–585, 2011.

[11] K. Goodhope et al. Building LinkedIn's real-time activity data pipeline. IEEE Data Eng. Bull, 35(2):33-45, 2012.

[12] Hadoop. URL: hadoop.apache.org as of 2014/5/1.

[13] C. Hewitt et al. A universal modular actor formalism for artificial intelligence. *IJCAI*, 1973.

[14] Y. He et al. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems. In Proc. of ICDE, pp. 1199–1208, 2011.

[15] HStreaming. URL: www.adello.com as of 2014/5/1.

[16] Impala. URL: www.cloudera.com as of 2014/5/1.

[17] J. Kreps et al. Kafka: A distributed messaging system for log processing. In Proc. of NetDB, 2011.

[18] K. Lee et al. Parallel data processing with MapReduce: a survey. SIGMOD Record 40(4):11-20, 2012.

[19] X. Liu. Data warehousing technologies for large-scale and right-time data. Dissertation, Aalborg University, DK, 2012.

[20] X. Liu, C. Thomsen, and T. B. Pedersen. ETLMR: a highly scalable dimensional ETL framework based on mapreduce. In Proc. of DaWak, pp. 96–11, 2011.

[21] X. Liu, C. Thomsen, and T. B. Pedersen. Mapreduce-based dimensional etl made easy. PVLDB, 5(12):1882–1885, 2012.

[22] X. Liu, C. Thomsen, and T. B. Pedersen. CloudETL: Scalable dimensional for Hive. In Proc. of IDEAS, 2014.

[23] N. Marz and J. Warren. Big data: principles and best practices of scalable realtime data systems. Manning, 2013.

[24] S. Melnik et al. Dremel: interactive analysis of web-scale datasets. *PVLDB* 3(1-2):330-339, 2010.

[25] L. Neumeyer et al. S4: distributed stream computing platform. In: *Proc. of ICDMW*, pp. 170–177, 2010.

[26] S4 distributed stream computing platform. URL: incubator.apache.org/s4 as of 2014/5/1.

[27] Scribe. URL: github.com/facebook/scribe as of 2014/5/1.

[28] Spark. URL: spark.incubator.apache.org as of 2014/5/1.

[29] M. Stonebraker et al. MapReduce and parallel DBMSs: friends or foes? Commun. ACM, 53(1):64–71, 2010.

[30] Storm. URL: storm-project.net as of 2014/5/1.

[31] C. Thomsen, T. B. Pedersen, and W. Lehner. RiTE: Providing on-demand data for right-time data warehousing. In Proc. of ICDE, pp. 456–465, 2008.

[32] M. Zaharia et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. NSDI, pp. 2–2, 2012.

[33] M. Zaharia et al. Discretized streams: an efficient programming model for large-scale stream processing. USENIX HotCloud, pp. 10–10, 2012.

[34] M. Zaharia et al. Discretized streams: fault-tolerant streaming computation at scale. In Proc. of SOSP, pp. 423–438, 2013.