Beyond MapReduce:
The Next Generation of Big Data Analytics

ET International, Inc.

Brian Hellig
ET International, Inc.
bhellig@etinternational.com

Rich Collier
ET International, Inc.
rcollier@etinternational.com

Stephen Turner
ET International, Inc.
sturner@etinternational.com

Long Zheng
University of Delaware
Lzheng.aizu@gmail.com

Abstract
ET International (ETI) is introducing a way to keep the best aspects of MapReduce and its popular implementation, Apache Hadoop, while reducing the number of add-on tools needed to make it relevant for commercial application. ETI’s novel multisource analytics product "HAMR" runs both batch and real-time streaming. It complements the current paradigm and accommodates the next generation of systems that will begin to render Hadoop MapReduce – as we know it – obsolete.

Given that Moore’s Law has an 18-month cycle time, it is urgent that information systems professionals budget and plan now for future generations.[6]

MapReduce does what it was intended to do: store and process large datasets in batches, inexpensively. However, many organizations that implemented Hadoop have experienced unexpected challenges, in part because they want to do more than MapReduce was designed to do. In response, an ecosystem emerged that continually introduces additional tools to help overcome some of those challenges. Unfortunately, this has only made implementations more complex and daunting, giving rise to the need for the simpler toolset offered by HAMR.

I INTRODUCTION
The name MapReduce originally referred to proprietary Google technology, but has come to generically mean a programming model used to store and process large scale datasets on commodity hardware clusters. One popular open-source implementation of MapReduce is Apache Hadoop.

The big idea behind MapReduce revolved around processing and analyzing big data. Although it was designed to "hide the complexities of scheduling, parallelization, failure handling, and computation distribution across a cluster of nodes"[7], Google developers pioneered the MapReduce model to handle really big data.[1] Dean and Ghemawat opened their breakthrough 2004 paper by suggesting that "MapReduce is a programming model and associated implementation for processing and generating large data sets."[1]

The MapReduce concept became extremely popular, in part because organizations could deploy it on available computing components for parallel processing. This allowed terabytes of data to be analyzed on vast networks of inexpensive computers. This commodity-based approach offered tremendous value and solved major problems for Google. Its search engine behemoth crawls, indexes, inspects and ranks the entire web.[3] MapReduce allowed them to aggregate information from a variety of sources, transform the data into different formats, assess application logs, and export the data for analysis.[4]

By 2007, Doug Cutting asserted "Yahoo! regularly uses Hadoop for research tasks to improve its products and services such as ranking functions, ad targeting, etc. There are also a few cases where data generated by Hadoop is directly used by products." He was proud to say, "Unlike
Google, Yahoo! has decided to develop Hadoop in an open and non-proprietary environment and the software is free for anyone to use and modify."[2]

This open source approach led to today's Hadoop 2.0 ecosystem and its variety of complementary technologies (such as Storm, HBase, Hive, Pig, Mahout and Zookeeper). Although these technologies fulfill different needs, they add complexity[5] because their capabilities go beyond what MapReduce was designed to do. Specifically, users want to extend the limitations of MapReduce’s current synchronous, batch processing of datasets.

As a solution to batch processing limitations and future-generation computing system challenges, ETI proposes the next generation of MapReduce: its novel product "HAMR," which uses asynchronous “Flowlets” and "Key/Value Stores" (patents pending) to allow processing multiple sources of data in real-time streaming or batch modes.

II. BATCH PROCESSING DATASETS

MapReduce is a synchronous approach to data processing, therefore it is synonymous with batch processing. Mapping generates a set of intermediate key/value pairs; reduce merges intermediate value associates with an intermediate key.[4] A key is a particular identifier for a unit or type of datum. The value can point to the whereabouts of that data, or it can actually be the identified data.[10] Dean and Ghemawat noted that their design was an abstraction that "allowed us to express the simple computations we were trying to perform but hides the messy details of parallelization." They built their model around sequential map and reduce basic building blocks in functional languages such as Lisp.[4]

MapReduce methodically batches processes enormous sets of data. In practice, large e-commerce sites must wait to see the results of log files analysis because MapReduce takes many hours or even days to process datasets measured in terabytes. (Large sites can have 100,000 visits per day, with each visit generating 100 log files). E-commerce sites that do not require rapid response could make adjustments daily or weekly if the analysis indicates a problem. For example, by analyzing shopping cart abandonment, they could identify the root cause and improve the customer experience. This is not something that needs to be done on the fly; multi-hour or overnight processing is fine.

However, other situations require rapid response. For example, a fraudulent credit card transaction in a retail store should be identified and stopped in no more than a few minutes, but typical batch processing can only detect the fraud hours after the merchandise has left the building.

The Hadoop Distributed File System (HDFS) contributes to the delay, because administrators must physically move the data from each system of record to HDFS. Even if moved hourly in smaller batches, this time-consuming process is neither appropriate for situations that require real-time streaming analytics (such as fraud) nor optimal for a range of processing algorithms.

III. NEXT GENERATION OF MAPREDUCE

ETI is introducing a new way to run both batch (when delays are acceptable) and streaming (when time is of the essence). ETI’s novel multisource analytics product "HAMR" is optimized to run in either batch or streaming mode and to reduce the number of add-on software tools found in Hadoop.

Although the terms "real-time" and "streaming" are often used interchangeably, they have different meanings. Real-time means that once an event happens, the results are returned within a "predictable" timeframe. In her article, Mary Ludloff used the example of a car’s antilock brakes as a real-time computing system, and the time in which the brakes must be released in a predictable timeframe.[8] In contrast, streaming relates more to user perception. That is, users with sufficient network bandwidth expect continuous output without noticeable delays.[9]

The option to run real-time streaming analytics is in increasing demand, but the network, hardware, and software systems must be designed to support this requirement. Database systems typically use structured query languages (SQL) or resource description frameworks (e.g., SPARQL) as their primary means of interfacing with the outside world. Although it is possible to express diverse queries in these languages, many operations, especially graph-related queries, require very complex syntactic constructs or
repeated queries. Furthermore, both database systems and more direct software approaches have a drawback: they perform acceptably only for particular problem classes. For example, neither database systems nor MapReduce easily perform a search within a graph.

As the industry merges heterogeneous data sources, it becomes necessary to interact with ad hoc relationships among data. Data scientists need ways to traverse these data structures in an efficient, parallel way. These data are too large to live on a single shared-memory system; they must be distributed across many systems. They can live in-memory for fast access, or reside on local disks suitable for larger volumes of data, but with less efficient access to the data.

Some distributed graph systems have been presented in the literature and market, such as Pregel and its derivatives (Giraph, GoldenOrb, JPreGel, Phoebus, Sedge, Angrapa, etc.), or federated/sharded share-nothing databases (BigData, Neo4j, etc.). However, there has been very little work on distributed graph systems that allow both more flexibility than Pregel (i.e., just storing temporary data in the vertex) and complex relationships among data stored on different computer nodes. Specifically, federated /sharded systems force internode joins to the application layer and limit the complexity of the joins and therefore, the data interrelatedness. ETI proposes – and is currently beta testing – a scalable system that will allow more flexible graph computations such as clustering, pattern searching, and predictive analytics on distributed heterogeneous systems – without forcing "shared-nothing" policies.

Such a fundamental transformation requires three software prerequisite features:

- **A runtime system** that manages:
  - Concurrency
  - Synchronization
  - Locality
  - Heterogeneous computing resource types such as different central processing units (CPUs), graphics processing units (GPUs), and field programming gate arrays
  - Distributed hierarchies of memory
  - Fault tolerance
- **A simple to use, domain-specific interface** to interact with this runtime system. The interface must provide methods to load, modify, traverse, and query a parallel distributed data structure—all without forcing the programmer to understand, or be concerned with, the concurrency, synchronization, or locality details of those operations.

- **Language constructs** that simplify and extend programmatic access through keywords and compiler analysis when the application programming interface is either clumsy or insufficient.

### IV Planning for Next Generation

Present-day big data software will be difficult to adapt to future-generation computing systems with many-core sockets and GPU accelerators, and their related, increasingly difficult programming, efficiency, heterogeneity, and scalability challenges. These challenges call for careful coordination between the very large number of executing CPU cores as they access highly non-uniform and widely distributed memories, which may incur large latencies for more remote accesses. Long-latency operations can seldom be predicted and scheduled statically to any great effect, partly because they may be highly data-dependent or complex, and partly due to unpredictable interference between hardware components. Making matters more difficult, the reliability of large-scale computing systems will likely be far lower than that of present-day systems at smaller scale. Future-generation systems applications will be required to handle and recover from what today would be considered major hardware faults in order to fully complete functions.

In any system with non-uniform or remote memories, maximizing collocation of data and computation minimizes the time and power overhead required for remote communications. Within a shared-memory domain, this is typically done with hardware caches. However, maintaining coherence among these caches requires an amount of communication proportional to the square of the number of caches. Thus, as the number of cores increases exponentially, the difficulty of efficiently maintaining cache coherency increases far more quickly.

Hardware vendors have several ways to solve this
problem, including lower-bus-traffic (but still fundamentally $O(n^2)$ for $n$ cache), coherence protocols, eliminating caches altogether, or moving the burden of coherence management into software. If software must handle some coherency issues and the runtime can address this, then the runtime might as well handle all coherency maintenance. Data movement costs time and power, but HAMR has several means to reduce those costs. HAMR takes full advantage of locality aware file systems, such as HDFS, by performing computation on the compute node where the data resides. In memory data sets are partitioned among the compute nodes where further processing can be performed. These partitions remain in memory until they exceed the available memory capacity, at which time they are intelligently spilled to disk. Programmers will see little to no impact on programmability as locality and memory management tasks are managed by the HAMR runtime.

Computing systems grow ever larger and more interconnected, and the boundaries between the systems introduce stricter limitations on the type of work that any of them can perform. Traditional parallel applications must explicitly partition their data and work, so that they can distribute it effectively to all available hardware resources. This is commonly seen in the practice of assigning work to nodes in a commodity cluster based on each one’s assigned ordinal rank.

However, as workloads become larger and less regular, it becomes increasingly difficult to create an effective static mapping from the work and data to the compute node. Virtualizing the way work is initiated and data is accessed can overcome this difficulty, by allowing processing and data to be decoupled from their initial locations and migrate as runtime needs dictate. By building a suitable execution environment and allowing running tasks to easily query their relative and absolute location within a parallel system, application software can also direct its own work/data movement – rather than leaving it entirely under the control of the runtime system.

V ASYNCHRONOUS FLOWLETS AND KEY/VALUE STORES

ETI's HAMR model uses a dataflow-based computing system. Although users still write map and reduce functions, HAMR treats these MapReduce tasks with a fine-grain and dataflow style parallelism, quite different from the coarse-grain, bulk synchronous parallelism found in Hadoop. In contrast to the existing MapReduce model, which starts with mapping and is followed by reduction, the HAMR approach allows users to implement multiple phases as needed. This new patent-pending workflow executes on a distributed computing system using a data parallel execution model. Each of the phases is a type of "Flowlet", and each Flowlet is a Resource, a Key/Value Store or a Transform.

ETI has coined the term "Flowlet" (patent-pending) to describe a data-flow actor in a workflow that performs a computation on an input dataset and produces output datasets. Flowlets can be flow controlled, for example to stop producing data when downstream actors are busy computing on other data and have no space to store the incoming data.

This is what makes the HAMR model scalable and well suited for future hardware upgrades in large scale systems. The system can process the flow control event from the downstream consumer actor, forcing the producer actor to stop producing output.

Downstream, data may be accessed from a variety of sources. These Resources may include social media feeds, sensor data, biometric readings, real-time market quotes or internal operational data. The Resource is responsible for converting the raw format into the structured, key/value format suitable for data parallel processing.

Key/Value Stores (patent pending) represent a reliable data structure that takes advantage of a paradigm already familiar to developers using the MapReduce model. These data structures exist across multiple nodes but are treated as a single data structure. A Key/Value Store manages its own memory, spilling to disk only when necessary. Read-and-write access to the data structures is based on key/value pairs, where the key represents an index and the value represents the object to store or retrieve. These data structures fall into three categories:

- **Distributed.** The data structure is partitioned among the nodes, and each node owns a unique subset of the key space.

- **Replicated.** The data structure is mirrored across all nodes. Each node can read the entire
key space, but write access is partitioned to a subset of the key space.

• **Shared.** The data structure is readable and writable by all nodes, but the physical memory for storing the values is partitioned among the nodes.

Within these three categories, several data structures can be implemented, including hash tables, distributed arrays, and trees. Key/Value Stores and Flowlets can be connected in a directed graph.

**Transform** is a common term in computer science, representing the conversion of one data format to another. If Key/Value Stores represent the data structures in a HAMR workflow, then Transforms represent the algorithms. Developers implement their application logic with a Transform, possibly interacting with Key/Value Stores.

In a traditional MapReduce model, the relationship between the map and reduce phases is fixed. That is what makes it synchronous: all outputs of mappers are pulled by the corresponding reducers.

However, with the HAMR approach, users have the flexibility to define the relationship among multiple Flowlets that can dynamically control the flow of data between the phases. The HAMR approach incorporates multiple operators to the application developer’s toolbox. These operators control iteration, streaming, real-time guarantees, and more. Operators are applied to a portion of the workflow graph. HAMR’s execution model enables these Flowlets to be executed in-memory in many cases, and is not restricted by the amount of cache memory on the system. Instead, with flow control and fine-grain parallelism, Resources, Transforms, and Key/Value Stores are aware of each others’ processing speed, so that they can adjust computing resources to each Flowlet, leading to higher compute utilization and proper load balancing. By implementing the HAMR approach within their Hadoop cluster, information technology departments will have one solution to manage, lessening the burden of continually creating and updating unique datasets for both batch and streaming analytics.

**VI. BENCHMARKING PERFORMANCE**

HAMR’s viability is best measured in performance benchmarks. University of Delaware graduate students and ETI performed these experiments, comparing HAMR to MapReduce. Four algorithms were run by the teams: K-Means, Word-Count, Graph Search and Clustering.

**K-MEANS.** The mean is used as a prototype at the center of a Voronoi cell, and other observations are clustered within that cell. Cluster analysis is commonly used at every stage in marketing, for segmenting the population and targeting each segment with different offers. In a comparison of performance on a K-Means clustering algorithm, the performance improvement was nearly 7x, in favor of HAMR.

![K-MEANS BENCHMARK](chart)

**WORD COUNT.** A fairly common algorithm determines the number of words in a file or web page. This method is being used more frequently in processing electronic medical records. MapReduce performs well, but HAMR increases the response time from 1.0 to 1.5x.

![WORD COUNT BENCHMARK](chart)

**GRAPH SEARCH.** This algorithm checks the values of all nodes in a graph. This is a complex algorithm and demonstrates the power of parallel processing. HAMR outperformed MapReduce by
nearly 30x. Graph Search can be used in logistics, to find multiple routes on a map.

**CLASSIFICATION**. Machine learning commonly involves a classification algorithm. A user can develop a hypothesis by selecting from classified units that conform to the theory. One type of classifier is Naïve Bayes. This is commonly used in financial services and actuarial science to assess probability.

HAMR outperforms MapReduce by more than 5x.

**VII. REFERENCES**


wisc.edu/~dusseau/Classes/CS739S06/Writeups/mapreduce.pdf.


http://searchenterprisedesktop.techtarget.com/definition/key-value-pair.