Big Data Analytics Survey

Daniel McCormick
Computer Science
Vanderbilt University
Nashville, USA
daniel.mccormick@vanderbilt.edu

Andrew Ragan
Computer Science
Vanderbilt University
Nashville, USA
andrew.s.ragan@vanderbilt.edu

Brandon Smith
Electrical Engineering
Vanderbilt University
Nashville, USA
michael.b.smith@vanderbilt.edu

Colin Hansen
Computer Science
Vanderbilt University
Nashville, USA
colin.b.hansen@vanderbilt.edu

***Abstract*— Big data technologies such as Hadoop, Spark, and cloud-based analytics provide significant advantages in the areas of storing, manipulating, and analyzing data. As the amount of data available continues to grow, however, technology needs continuous updating and new techniques must emerge. In this paper, we provide a detailed survey of work done in this area. Subsequently, we attempt to provide a taxonomy of these works by dividing them into 4 key areas: reducing costs associated with data transfer, efficiently and effectively allocating resources, meeting usability standards and user expectations, and managing the massive data required to justify these systems. Next, we provide some open problems for research before concluding with a brief review. Through this paper, we hope to provide a good entry point for people new to the area of big data analytics as well as a good refresher of key topics for those already familiar with this area of study.**

Keywords—Big Data, Analytics, Data Management

I. Introduction/Motivation

Big data technologies such as Hadoop and cloud-based analytics bring significant cost advantages when it comes to storing large amounts of data. With the speed of systems like Hadoop and developed analytical technologies such as Spark, combined with the ability to analyze new sources of data, we are able to analyze information immediately and make decisions based on what they have learned. Big data analytics is an important aspect of business and research, and so it is the topic of focus for many researchers.

Fig. 1. Value of Big Data Analytics [21]

While there are many positive growths in the area of big data analytics, the amount of data around us is growing rapidly and it is a challenge to keep up. Many of the existing systems and techniques currently in place are outdated or inefficient and are in desperate need of improvement. Here, we discuss the important strides made to improve the field of big data analytics, both focusing on strategies used to create successful systems and successful systems themselves. In addition, we provide a taxonomy that outlines 4 prominent themes in recent work: reducing costs associated with data transfer, efficiently and effectively allocating resources, meeting usability standards and user expectations, and managing the massive data required to justify these systems. Finally, we finish with some of the open problems remaining in the area and a brief summary of the paper.

II. Survey of Related Work

For the purpose allowing for easily deployed analytic models, one group set its sights on Vertica and Spark. Vertica is an enterprise database that provides robustness for analytics of mission-critical data, and Spark is an open source big data computation engine that allows machine-learning models to be deployed on Vertica data. The two main challenges overcome by this work were transferring data quickly, reliably, and robustly to support a pipeline for Spark to use Vertica as a fast analytic data source and to support spark as an extract transform-load (ETL) engine for Vertica and creating and then deploying Spark ML models within Vertica for in-databases scoring. Transferring data from Vertica to Spark (V2S) utilizes Vertica's data layout and its ACID semantics to enable fast data transfer to load a consistent view of the data along with exactly-once semantics. Transferring data from Spark to Vertica (S2V) again uses Vertica's ACID properties to guarantee exactly-once semantics, but to save data. Vertica is used as a log of Spark's MapReduce batch processing progress and final status. A series of tables (including the task status table) and phases are used to generate the target table that will be written to the Vertica database. They have also developed APIs for deploying and reading Predictive Model Markup Language (PMML) models to/from Vertica. This allows users to run predictions inside the database using the scoring user-defined functions in an SQL query. This allows prediction models to be trained in Spark using Vertica data and then deploy the model into Vertica to do predictions on stored data. [1]

Data-intensive computation (DISC) frameworks are shared by multiple entities in an organization and the resource

manager is responsible for allocation of compute slots among jobs such that service-level objectives (SLO) are met. Amoeba is a system developed to address this that enables lightweight elasticity in DISC frameworks. It leverages the property that it is possible to judiciously split the original work assigned to tasks without altering the result of the overall computation. Amoeba identifies such safe points to split a task, which also obviates carrying over of state across the splitting. Using this low-overhead splitting, Amoeba resolves contentions for slots by safely exiting a running task, committing its output and spawning a new task for its remaining work. Amoeba achieves this by keeping a memento for task progress via periodic updates. With such elastic resource consumption by jobs, the resource manager can over-allocate slots to jobs without missing its SLOs or wasting computation. The same mechanisms used in Amoeba for scaling down the slots assigned to a job can also scale up: as slots become available, a running task can be safely terminated and multiple new tasks can be spawned for the remaining work. [2]

To take on the issues surrounding wide-area big data analytics, Vulimiri et al. focuses on minimizing bandwidth over geo-distributed data structured as SQL tables, a dominant paradigm. They support the entire array of SQL operators on global data including Joins, providing exact answers. The four main contribution here are jointly optimizing query execution plans and data replication to minimize bandwidth cost, a technique that modifies query execution to collect accurate data transfer measurements, caching all intermediate results to eliminate data transfer redundancy using deltas, and demonstrated gains (Geode) which is built on top of Hive. Geode achieves 250x data-transfer reduction over centralized approach and 360x improvements other various scenarios and benchmarks. Geode constructs a distributed plan for queries in two stages. First, choose join order and strategies utilizing a customized version of Apache Calcite (Calcite++) to construct optimal join ordering for the query. Then choose a distributed join algorithm for each join; schedule tasks by assigning tasks to data centers while taking into account task input dependencies and bas data regulatory constraints. [3]

Nicolae, Costa, Misale, Katrinis, and Park discuss methods of shuffling data between nodes. They begin by motivating their work by pointing out that data shuffling is very important and can contribute significantly to the runtime of algorithms in many distributed models (e.g. Apache Spark). They began to talk about some basic principles and challenges in the area. For example, they discuss principles like data-locality centered design, which is the principle that the storage layer should be co-located with the computational elements. Many in-memory systems like Spark try to minimize interaction with the storage layer, reducing disk I/O bottleneck, but this is challenging to do across many processors. Another challenge of data shuffling include that it requires many-to-many communication across the network. This can be addressed with asynchronous communication, but this has the potential to blow up the memory requirements due to buffers. Other challenges include that there are huge amounts of data in large systems and there will be huge network and memory strains. This tradeoff needs to be balanced – systems want to accumulate as many data blocks in background as possible to limit I/O bottleneck, but want to keep in-flight reducer limit low to reduce memory usage. From here, they began to provide recommendations to address this basic problem – given processes, each with a local dataset D, where separate parts of D are meant to be accessed by different processes how do you best request blocks to be sent across the network. To approach this problem, they generated a strategy with several recommendations. First, in order to choose which nodes to request blocks from, the requester node should coordinate all requesting processes to keep track of the total amount of data processed and prioritize the process with minimal amounts of data coming in. In addition, requesters should prioritize nodes that have spent the longest amount of time sending each data block on average to ensure that those blocks can be sent and do not end up being bottlenecks. At the start, when this information is unavailable, nodes should prefer remote nodes that immediately follow them in some circular ordering - this balances the requests out. Next, in order to choose how many blocks to request from a node, several key concerns must be kept in mind, most notably that issuing a request per block is expensive, so systems want to maximize number of blocks they get back per request. This means that the system must trade off requesting small amounts as needed which will be less efficiently transported but consume less memory with requesting large amounts that consume a lot of memory but ensure the node won't run out of blocks. To balance this, the authors suggest defining some minimum size for data and only requesting data amounts larger than this minimum. They do, however, recommend allowing this limit to be fluid and go down when the need for more blocks is higher and vice versa. [13]

Leyshock et al. present a paper on Agrios: a hybrid system compised of R and SciDB. Since R is used for vector and matrix analysis in a single threaded fashion and SciDB is designed for analyzing very large data sets, the two would not seem to be compatible in the switching of data between them. Agrios's goal is to minimize data movement between R and SciDB by calculating the cost of each move in a staging phase. Agrios determines what best steps there are to take between the processes based off the lowest cost formula. Different transformations occur on the data: both consolidating and reductive transformations, which take time to occur but ultimately save the system its use in resources. [16]

A system that schedules flow in a network well is that of Hedera. The premise of cloud-based operations is difficult to manage due to unknown workloads and the requirement of high bandwidth. Hedera collects information in these operations pertaining to the flow of data and computes "non-conflicting paths" for the data to flow. This utilizes a technique known as multipathing. Multipathing in its current form causes many collisions, but Hedera improves this method. To do so, Hedera makes a guess at where it thinks the might best go in the network, and then refines its guess by assigning certain switches to particular hosts, ensuring that other hosts can also get the data without unnecessary collisions. [18]

Hua et. al. present a technique called FAST that allows data to be processed by extracting feature vectors via sifting. This work is motivated by the vast amount of data that must be searched when a query is made. Due to the great amount of information, the latency is high, which is often unacceptable to

the user. These long latencies thus diminish the value of the analytics since the data arrives slower than what is needed. FAST is based off three principles that can be applied to a wide variety of inputs. The first is semantic aggregation, grouping inputs using a Bloom-filter. Secondly, Locality-Sensitive Hashing (LSH) based hash tables are summarized in a way that can be utilized by the third principle: Flat Addressing. The data is made readily available to the user by means of flat addressing which allows the information to be accessed in O(1) time. [19]

Islam et al. use hardware tactics to increase the performance of the distributed file system Hadoop. The authors present an argument for the use of non-volatile memory, which has been growing in popularity in recent years due to its "midpoint" between SRAM and SSD memory on the price and speed scale. The system they design is call NVFS: Non Volatile Memory (NVM)- and Remote Direct Memory Access (RDMA)- aware Hadoop Distributed File System(HDFS). RDMA is a communication scheme that allows the process of writing to a HDFS node to be done in near parallel, increasing the speed at which data arrives at a DataNode (a type of node that manages data in HDFS) much faster. This communication method alongside the increase in memory access speed from the NVM allows the NVFS to have great improvements over traditional HDFS. The authors designed different types of block structures to organize the data in each DataNode, buffering the writes in different schemes of NVM. Compared to a flat architecture, the block architecture increased the read speed, which is the primary advertised improvement of this system. [17]

Camdoop is another system that aims to reduce the bandwidth required to perform big data analytics. Many big data tasks use a highly scalable model called the partition/aggregate model. This splits an input dataset over many servers, which then run some functionality to produce an intermediate result. These results are then aggregated to generate the final result. This aggregation is often also done across many servers, but this requires intermediate results to be mapped to the appropriate server in what is called the shuffle phase. The cost of this step depends directly on the size of the intermediate data. However, it is often observed that such a function's final result is far, far smaller than the size of the intermediate result. Camdoop tries to combat this problem by combining the popular MapReduce algorithm with the CamCube platform, a cluster topology that removes the distinction between the physical and logical network to allow for custom routing protocols. Camdoop takes advantage of this to drastically reduce the network traffic present in these programs by making it a function of the result's size rather than the amount of intermediate data. Camdoop also pushes the work done in this step from the edge into the network itself to further offset the cost. The system is also tolerant to link and server failures due to the high tolerance CamCube affords. Even outside of its primary use cases, this architecture allows for optimizations over standard MapReduce implementations. [8]

Answering the lack of research surrounding replication over networks, Sinbad is presented, a system that leverages the flexibility in endpoint placement during distributed writes to steer replication transfers away from network hotspots. Network-balanced placement has two implications. First, it improves cluster file system (CFS) write throughput by avoiding network contention; response times of tasks that write improve as well. Second, by steering CFS' traffic away from hotspots, throughput of non-CFS traffic on those links increase. In data-intensive clusters, this speeds up tasks that shuffle intermediate data and jobs with large shuffles. Sinbad is a measurement-based system to perform network-balanced replica placement during distributed writes. Given a replica placement request –with information about the location of the writer, size of the block, and the replication factor –Sinbad must return a set of locations for the CFS master to replicate that block to. All information about a block request is unknown prior to its arrival. Instead of retrieving responses from the best sources, we have to replicate large blocks to multiple machines in different fault domains without introducing significant storage imbalance across the cluster. The problem of placing replicas to minimize the imbalance across bottleneck links is NP-hard even in the offline case. Sinbad employs algorithms developed by exploiting observations from real-world clusters to perform reasonably well in realistic settings. Sinbad is designed to replace the default replica placement policy in existing CFSes. Similar to its target CFSes, Sinbad uses a centralized architecture to use global knowledge while making its decisions. Sinbad master is collocated with the CFS master, and it makes placement decisions based on information collected by its slaves. Upon the heartbeat of the CFS slaves, Sinbad slaves report back information including incoming and outgoing link utilizations at host NICs and current disk usage. Sinbad master aggregates the collected information and estimates current utilizations of bottleneck links. Sinbad uses host-based application layer techniques for measurements. Since Sinbad agents are collocated with that of the parent CFS, host failures that can take Sinbad slaves offline will take down corresponding CFS agents as well. If Sinbad master dies, the CFS master can always fall back to the default placement policy. Because most CFSes already have a centralized master with slaves periodically reporting to it –Sinbad does not introduce new scalability concerns. Furthermore, piggybacked measurement updates from Sinbad slaves introduce little overhead. [4]

Speed is not the only factor to consider in data replication. Many cloud-computing systems use data replication without taking into account the quality-of-service (QoS) requirements of the application, because cloud-computing systems take advantage of the heterogeneous node structure. As a result, the data from a high-QoS application may have a replica in a node with low performance. This can cause issues if the high-QoS node fails, and the user is left with just the low-QoS node. Lin et al. discuss two algorithms to combat this problem. The first algorithm, called high-QoS first-replication (HQFR), is perhaps the most intuitive approach to this problem. Essentially, you replicate the data from higher QoS requirement applications first so that high QoS nodes are backed up in other high QoS nodes. While this approach is intuitive to program and fast, it does not achieve any sort of optimal solution. The second algorithm is more complex, but achieves the optimal solution, where optimal means minimizing replication cost and QoS violations. This is done

by mapping the problem to the minimum-cost maximum flow problem, which can be solved in polynomial time. Again, this algorithm has the clear advantage over HQFR in that it finds the optimal solution, but it comes with additional complexity in terms of both programming and runtime. However, both strongly outperform QoS-blind methods, so both provide the user with more a more reliable service. [9]

Xuana et al. address issues surrounding data storage and I/O access in computer clusters. Usually the choice is between suing the high performance computing (HPC) cluster's parallel file systems or using a distributed file system such as HDFS on compute nodes. The HPC file system has the advantage of high capacity and low-cost fault tolerance at the cost of poor performance limited by network and disk I/O bandwidth of the storage nodes. HDFS has high aggregate I/O throughput but suffers from costly data fault tolerance and low storage capacity. This paper proposes a two-level storage system that attempts to gain the best of both worlds. The two-level storage system combines an in-memory file system on the compute nodes and a parallel file system on the data nodes. The implementation uses Tachyon as the in-memory filesystem and OrangeFS as the parallel file system. Although running Hadoop on Tachyon alone can also take advantage of high I/O throughput and data locality, it has two issues. First, the capacity of Tachyon is limited compared to large storage capacity on data nodes. Second, Tachyon uses lineage to recover data when there is a fault. This recovery incurs computing cost. In the two-level storage, local data always has a copy in OrangeFS; thus, OrangeFS provides fault-tolerance for Tachyon. [5]

GRASS uses a speculation algorithm to reduce the number of "stragglers" in approximation jobs. Approximation jobs are important because they handle big jobs that would take very long to process and run an algorithm until an approximation within a certain bound is reached. Since jobs are based off speculation, it is undoubtable that some will "straggle" behind due to over-speculation of time. The challenge addressed in this paper is solved using two types of speculation: Greedy Speculation (GS) and Resource Aware Speculation (RAS). GRASS addressed two types of bound approximation jobs: deadline-bound (jobs that must complete within a certain window of time) and error-bound (jobs that must be within a certain region of error to be of meaning). GS is good for jobs with few waves of computation, whereas RAS is good for those jobs with many waves of computation. GRASS utilizes both GS and RAS to maximize the utility of time and resources. To do so, it determines a "switching point" at which the job (starting initially in RAS, as if to run for many waves) switches to GS (where the algorithm has determined that the number of waves left is relatively small). [20]

Another factor of big data analytics that affects performance is hardware configuration. Many big data analytics tools use virtual machines or other methods that provide various choices in hardware configuration. Venkataraman et al. developed Ernest, a performance prediction framework for big data analytics used to predict the performance of tasks on various configurations in order to select the optimal configuration. Of course, this prediction must come with low overhead in order to be valuable. To achieve this, Ernest uses a system modeling approach, which has the advantage of only requiring a small amount of training data. Specifically, Ernest uses only two features about its samples: scale (how much of the original dataset it represents) and the number of machines used for execution. Ernest then uses a formula that represents different costs of a job, including serial computation time, parallel computation time, and communication costs. Ernest then creates a model by fitting the dataset to specific values for this formula. Then, the framework uses a technique called *experiment design* to choose only a small amount of the original dataset for training data. The model is tested against this data using a method called cross-validation, where if there are *m* training data points, the model is tested *m* times, each time against every data point but one. If a model does not seem to fit its data, Ernest can extend the model to look for additional features for a more accurate model. The paper then evaluates Ernest based on running various jobs on Amazon EC2. This shows that on long running jobs, Ernest predicts the correct runtime within 20% error while only incurring 5% runtime overhead. The use of experiment design was shown to improve accuracy by 30-50%, and extending the model helped increase accuracy in cases such as very sparse datasets. The ability to predict performance with high accuracy and low cost can be a very helpful tool in maximizing resource efficiency. [6]

Most of the works discussed so far focus on the network, disk I/O, or straggler tasks as the bottlenecks to widen in order to speed up performance. Each of these ideas focus on a single component without forming a comprehensive approach to improving end-to-end performance. Ousterhout et al. formulate a method to pinpoint bottlenecks, and to use this method to show that these assumptions about common bottlenecks are not necessarily true. The paper uses a method called blocked time analysis, which quantifies how much faster a job would complete if there were no blocking on the disk or network. This allows us to measure the best-case runtime after making an optimization to the disk or network. This method showed that, with no blocking from disk I/O, the median improvement would be 19%, that the median improvement with no blocking on the network was only 2%, and that without any extra time spent on straggler tasks, the median improvement was only 5-10%. Additionally, the paper went on to pinpoint the causes of the stragglers they encountered, challenging the common idea that these stragglers exist for indeterminate reasons. The idea of the paper was not to show that efforts shouldn't be focused on increasing network or disk speed, but rather that bottlenecks can and will change over time, and that analyses such as blocked time analysis should be used to remain aware of what the bottlenecks truly are. This way, both programmers and researchers alike can focus their time on improving performance where it is most impactful. [7]

Zhang, Wang, Tewari, Schmidt, and Kakrania attempt to provide a detailed layout of the medical imaging processing pipeline and bottlenecks. As with most big data, they identify five key attributes associated with the data: volume, velocity, variety, veracity, and value. They also describe the process of medical image analysis in the context of multi-atlas segmentation, or dividing images of the body into body parts. This is currently mostly done with Spark. They observe that

memory is the main bottleneck, followed by CPU. Because of this, they recommend balanced job scheduling across nodes and a distributed memory-sharing layer to alleviate memory costs. They also point out that increasing CPU is the next step if the memory bottleneck can be dealt with. They also argue that shared distributed storage should be preferred to shared-nothing storage since data locality makes things harder and does not help performance because I/O is insignificant. Overall, they point out that muti-atlas segmentation does not parallelize well to Spark and customized systems should be used to optimize performance. [14]

Nothaft et al. attempt to identify the key components of scientific analytics systems, provide some recommendations, and implement it in a custom system. First, though, they identify the shortcomings with some current systems. For example, the current state of the art genomics pipeline uses a single heavy node and does not allow for scaling of data. Further, while some map-reduce has been used, this does not mesh well with the legacy data formats and the whole pipeline is somewhat of a mess. On top of this, this problem is not unique – many scientific systems lack the structure necessary to provide efficient processing of large amounts of data, yet they are often asked to perform this type of processing. Because of these issues, Nothaft et. al. endeavor to create a standardized system that can be applied to any scientific application. In the process, they identify several key components. First, scientific systems should have clear layering. In fact, they identify seven clear layers: physical storage, data distribution, materialized data, data schema, evidence access, presentation, and applications. While many systems allow these layers to become muddied, the authors argue that these layers should be kept separate. Next, the authors identify that scientific workloads tend to be associated with coordinates, that scientific data tends to need to be sliced into many different views, that scientific applications tend to rely on statistic methods, and that scientific applications are usually data parallel. Using these observations and some others, the authors create a standardized stack that can be used across scientific domains and apply it for a single system used for genomics data pipelining called ADAM. Within this system, they included several key attributes. First, they emphasize the importance of data independence. Second, they talk about pushing computation to data rather than bringing data to the computation nodes. Third, they use a denormalized schema to provide O(1) parallel access to data. Finally, they emphasize the standardized stack with clear, well-defined interactions between layers of the stack. [11]

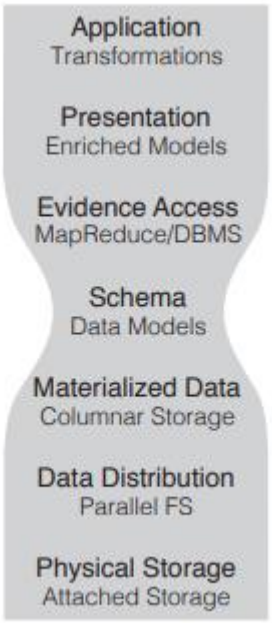

Fig. 2. Standardized stack for scientific computing.

Melnik et al. describe Dremel, their scalable and interactive query system for read-only data. The goal of the system is to provide almost instantaneous responses to small to medium sized queries over large datasets, scaling to 1000's of CPUs and petabytes of data. The key requirements of the system, therefore, are that it must be interactive, highly parallel, and with a very flexible data model for different types of data. One of the key takeaways of Dremel is that it uses nested columnar storage. Columnar storage is a paradigm that stores features together rather than instances together. This allows queries that only require a couple features to be run very quickly but queries that require most or all of each instance's features tend to run very slowly since each instance has to be reconstructed. It uses Google File System to store all of this data.

There are several key challenges with columnar storage that needed to be overcome. First, ensuring that storage is lossless is challenging. To overcome this, they use a special field to encode repeated values and null values so that they can be accounted for. Next, quickly encoding instances is challenging. To account for this, they use a recursive system that goes deeper into each record. Finally, efficient record assembly is challenging since the records need to be reconstructed across different columns. To address this, they use a bunch of readers (one per column) and have one central machine deciding which reader to look at next. This also allows it to ignore unused columns, a huge gain. For the user interface, the system uses a SQL-like language that is parsed natively with a tree-based implementation. The query is propagated down the tree and results are passed back up and aggregated at every node. This is clearly faster if the operations are commutative and associative so that they can be performed at each level. In addition, when allowable, slow leaves are ignored and only a certain percentage (e.g. 98%) of the data is read to speed up the query execution. Overall, this performs much better than traditional systems like MapReduce on small to medium sized

queries, but it unsurprisingly performs much worse on large queries. [15]

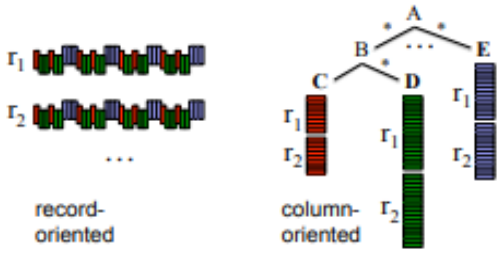

Fig. 3. Record-oriented vs column oriented storage.

Gray et al. attempt to provide a look at how scientific data management should evolve over the next decade or so. The paper begins by motivating the need of an evolution of techniques. Specifically, it talks about how demand for tools and computational resources is rising faster than data volume and how IO bandwidth is not growing as fast as storage capacity. From here, the authors begin to examine a wide set of areas and provide recommendations or each of them. First, they talk about data centers.

Here they make two key points: computation should be moved towards data in data centers rather than moving the data itself and data centers should be self-healing with replication. Next, they discuss metadata. They place a large emphasis on the maintenance of useful metadata and argue that it should be standardized in format, units, and method of publication. They also argue for data independence, with both metadata and other data. They also draw a distinction between physical data independence, where the program does not need to know where the data is stored just how to access it, and logical data independence, where the program doesn't need to know how the data is structured just how to access it, though they do emphasize the importance of both types of independence. Next, they discuss the need for set oriented data. This data is organized in sets so that order of access does not matter rather than in arrays where order does matter so that it can be accessed in parallel. Finally, they end their paper by recommending that scientific data management systems be moved to database management systems. They argue that the advantages of providing replication of data in different forms (materialized views) to provide fault tolerance and faster access at the same time, providing physical and logical data independence, and providing parallel access and search by value rather than location makes database management systems the right systems for most scientific applications. They do, however, recognize one major shortcoming: the mismatch between programming model and database capabilities. Because of this, they recommend a movement towards databases integrated with programming languages. [12]

While speeding up big data jobs is certainly important for productivity, metadata, which provides information about the data itself, plays a vital role in the actual analysis of large datasets. However, much of the most important metadata is often generated in a domain-specific manner, which can be very time consuming and inefficient and lead to even more inefficiency when combining data from different domains. Other important metadata can be lost under certain workflow systems. Blanas & Byna propose an integrated metadata management system that will address these problems. This system would allow for semantically rich metadata that would augment datasets. This metadata would be stored within the dataset itself, and would be accessible via the same interfaces as the actual data to avoid overhauling current data access interfaces. The system would generate, transfer, and manage all of this metadata automatically as a part of the standard data management. By maintaining metadata generated by previous analyses, reuse of a dataset is made much simpler. A scientist will have access to summaries of these analyses. These summaries from previous analyses can help a scientist quickly become acquainted with a dataset without having to analyze the data themselves. Hardware configuration and performance details from previous analyses can help scientists develop their own analyses with less debugging. This will all be stored with the data when it is curated, so the even scientists or the public can benefit from this information if they ever need to. Certain fields such as plasma physics, climate modeling, and neuroscience have already expressed a direct need for such information-rich metadata. [10]

Now we discuss our taxonomy of these recent works. We have reduced the advances made to four motivations: reducing costs associated with data transfer, efficiently and effectively allocating resources, meeting usability standards and user expectations, and managing the massive data required to justify these systems.

III. Cost of Data Transfer

Obviously due to the sheer size of data required for big data analytics, datasets cannot be stored in one place nor can processing occur in one location. This is why there has been plenty of research focused on reducing the costs of moving data where. Camdoop, Sinbad, and Geode are results of attempts to reduce network traffic and thus speed up the flow of information. Camdoop exploits the ability of custom forwarding and processing of packets on path to perform in-network aggregation improving performance in the shuffle and reduce phase of MapReduce-like jobs. Sinbad leverages the flexibility in endpoint placement during distributed writes to steer replication transfers away from network hotspots. Geode Providing wide area analytics while minimizing the needed bandwidth over geo-distributed data structured as SQL tables and supporting all SQL operators.

Other works have focused on reducing costs within systems and between nodes. One method developed consists of the Hadoop Distributed File System (HDFS) that utilizes Non-Volatile Memory for Remote Direct Memory Access communication with the file system that shows a great improvement over standard Hadoop. Another system was designed to choose which nodes to receive blocks from and how many blocks to request. This ensures no processes on a node are starved of the data blocks needed to proceed while also not exceeding a threshold on memory. A two-level storage system was also developed using a HDFS that utilizes Tachyon as an in-memory file system on compute nodes and OrangeFS as a parallel file system on data nodes resulting in high I/O throughput and data locality with low-cost fault tolerance.

Lastly in an attempt to reduce the costs of transferring data and models between a database and an analytical engine, a method was developed for enabling fast and reliable transfer between Vertica, an enterprise database, and Spark, an open source big data computation engine allowing for models to be built in Spark using data from Vertica and for models from Spark to be deployed within Vertica to produce useful analytics.

IV. Resource allocation

Many factors motivate the careful allocation of resources when analyzing large data sets. Due to the sheer size and processing time of the data, all resources must be used in the most efficient way possible in order to enrich the user experience (or in some cases meet user expectations). The complexity surrounding the resources and their management makes this task a non-trivial one. In order to identify areas of improvement in any system, one must identify the bottlenecks that slow down performance. This, along with predictive models that can determine what the future causes of slowdown might be in a system, provides valuable knowledge that can be used by resource managers in order to better utilize the system resources.

There have been some attempts to create a machine-learning algorithm to sample a dataset before it is processed. Advantages to a system such as Ernest [6] would be that a model would be chosen preemptively to best fit the nature of the data to be processed. If a learning algorithm could be developed to do this accurately every time, this would be the preferred method of choosing models for data. However, reality is not as ideal: mistakes in the learning algorithm can still be made, leading a dataset to the wrong model. This would result in an inefficient use of resources. Ernest has shown a way around this, attempting to learn from its mistakes and better determine the correct model which to assign a similar set of data. [6]

Similar methods have been explored in the determinations of bottlenecks in network and CPU tasks. In Ousterhout's paper [7], analysis of tasks with and without blocking pinpointed certain failure points in the system. If a user could know which has more bottlenecks (the CPU or the network), they might be able to optimize the system to better accept and process data. Perhaps the style of system presented by Hedera [18] could be utilized, since it manages the flow of data to hosts in a network by allocating specific switches to hosts rather than flows of information. This allows all other users to receive information even if a switch or portion of the network goes down. If information is getting consistently lost in a network, its resources should be examined and reallocated to best mitigate slowdown and straggler tasks.

V. Usability and User Expectations

Much of the research in the field of big data analytics focuses on speed or efficiency. It is important to note, however, that many applications have a user-facing aspect as well. Cloud computing tools such as Hadoop store large amounts of data from a variety of users with different uses and expectations. The user-facing issues that we found in our research primarily fall under two categories: balancing the needs of multiple users with different tasks and expectations, and providing responsiveness to users who may need to perform multiple tasks on the data.

The most obvious issue that arises with having multiple users with many different tasks is balancing these tasks in a fair way. While this may seem like your ordinary scheduling problem, the large amount of data in big data analytics can make this much more complex. For example, saving state for context switching can be much more expensive than for programs with less state data. Amoeba, as discussed earlier in the paper, is a system used to help alleviate this problem in DISC frameworks. Amoeba takes advantage of the divisible nature of many big data analytics tasks, marking safe points in tasks that allow for low-overhead splitting into subtasks to run later [2]. Since datasets in big data analytics are so large, it is often sufficient to run the job only on a subset of the data. How we choose this subset can be important. Approximation algorithms can help perform a job while guaranteeing a certain degree of accuracy without having to use the entire dataset.

However, these algorithms often have the issue of 'straggler tasks,' certain subtasks that lag behind the rest, reducing the advantage of these approximation algorithms. Algorithms such as GRASS, which we found in our research, can optimize these approximation algorithms to reduce the impact of these straggler tasks [20]. Solutions such as these pinpoint common properties of big data analytics jobs such as size and divisibility in order to maintain the speed and accuracy even when scaled to many users.

Multiple users may also have different expectations. One issue that arises in all distributed systems is reliability. In order to live up to a certain standard of reliability, data in distributed systems is often replicated in multiple nodes to allow for a certain number of failures. In cloud-based systems, however, nodes are associated with a certain quality of service (QoS), so data from a high QoS node may be replicated in that of a low QoS node. In one paper, Lin et al. discuss two algorithms to tackle this problem: a greedy algorithm and an optimal one, with the greedy being faster but obviously performing worse than the optimal algorithm [9]. Tradeoffs such as these appear throughout the field of computer science, and they are particularly important to consider on a scale as large as that of big data analytics.

Another issue that arises with big data applications with a user-facing side is responsiveness and latency. With such a large amount of data, a task's runtime and latency is at its worst. While a wait is unavoidable for complex operations, many users use the data for relatively simple uses. These simple operations, particularly if they are known, should be fast. Dremel [15] and FAST [19] both discuss architectures for storing data that is meant only to be queried and is therefore read-only. By optimizing for specific use cases, we can speed up operations for the users and provide a responsive application.

User-facing big data applications introduce their own problems into an already saturated population. In many cases, researchers have been able to apply methods used in other

problems, while other cases require a more user-oriented approach.

VI. Data and results Management

Data and results management is obviously an incredibly part of the pipeline. While this is sometimes an overlooked part of the pipeline, it can account for a large part of the computation time [12] and can often effect how helpful the output itself is. While there are many important principles that guide data management, there are several key principles that should guide data management.

First, there should be a clear separation of components. One of the things that Nothaft et. al. recognize in their paper is that many systems, especially those used for scientific management, lack clear and well defined stacks. They argue that computation should be separated into clear and well-defined layers. While this may prevent engineers from taking advantage of certain optimizations in their code, that time will be more than made up for in most cases by the clarity of what is going on as well as the reusability. Often software stacks that are breached can become almost unrecognizable. Because of this, they recommend the following 7 layers be used for scientific data management systems: physical storage, data distribution, materialized data, data schema, evidence access, presentation, and applications. [11] The division of these systems into clear layers cannot be understated and the APIs between these systems should be clear and well defined. Gray et. al. also argue for the importance of separation of components, though they argue for it specifically in the context of data independence. In particular, they argue for both logical and physical data independence. [12] This allows for much easier development and easier to maintain and reuse code.

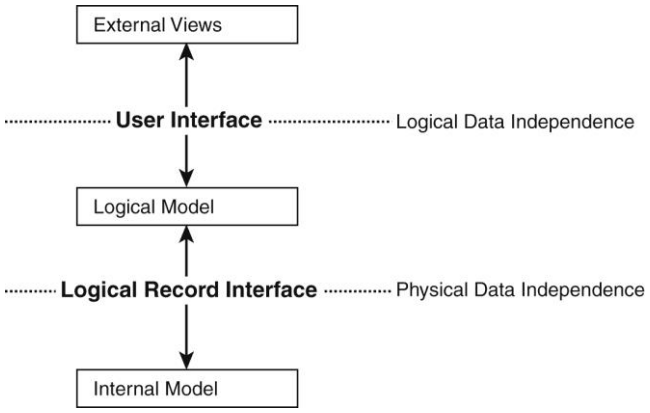

Fig. 4. Data independence illustrated [22]

Next, the maintenance of clear and standardized metadata is vital for effective data and results management. Gray et al. specifically argue for the standardization of not just metadata and its units, but also for its management which they argue should be automated [12]. This provides useful information to anyone using the system and allows for easier adoption and transferal to new systems. Blanas and Byna go even further, providing a clear description of what should be contained in the metadata. Specifically, they argue that it should contain the identity of the dataset, dataset descriptions, performance information, relationships between datasets or tasks, and user-defined metadata. They also argue that metadata should be stored in the dataset itself. [10] This again allows for easier adoption and usage. It also provides some measure of data independence as the user does not need to know how the metadata is stored, just how to access it in the dataset.

Finally, domain specific optimizations should be put in place to speed up data and results management. For example, Melnik et al. describe a system, Dremel, which prioritizes fast, read-only queries with response times below 5s. Because of this, it uses read only data, column stores, and tree based execution to provide really fast results for small queries [15]. This might not be practical for other systems, but the domain specific goals have to be taken into account. On the other hand, in describing systems for medical image analysis, Zhang et. al. argue for balanced job scheduling and distributed memory-sharing layer since memory is the main bottleneck. [14] These completely different conclusions are indicative of different requirements and the requirements of the systems must always be taken into account when designing their methods of managing data and results.

VII. Open Problems

Regarding data shuffling, Nicolae et al. have thoroughly explored methods for efficiently shuffling data for a single task. [13] There still exists, however, significant exploration to be done on optimizing shuffling techniques for concurrent tasks that are both trying to shuffle data. There also remains work to be done on shuffling data while trying to minimize interference with other workloads (whether they are using Spark or not).

In addition, while Nothaft et al. examine standardized systems for scientific analytics systems, currently many of these problems are treated differently than their system currently operates. In particular, many problems are posed as graph problems and their system is not optimized for this case [11]. Because of this, there is significant room for research into a similar system that is standardized for scientific graph applications.

As of the publication of the paper by Lin et al. regarding quality of service aware data replication algorithms, no cloud computing system had actually implemented these algorithms. Further research would be to determine if such a system has since been implemented. If so, it would be helpful to see how impactful the algorithms were on maintaining quality of service in a real use case. If not, implementing such a cloud computing system and analyzing the results would be of great interest. [8]

Venkataraman et al. pose a few problems regarding their performance prediction framework, Ernest. Namely, they would like to analyze how certain statistical properties change with hardware properties such as network latency. Additionally, in the benchmarks of the framework, they note that there are key metrics that seem to affect performance, and they intend to integrate those metrics into the features used in the machine-learning algorithm used by Ernest. [6]

As for a metadata management system, Blanas & Byna have established important properties and concepts to be in their proposed system, but also point out many open problems that must be solved before full implementation of such a system. Should the framework exist on the application layer, library level, or even down in file format or OS levels? Each has its benefits and its drawbacks. How can we discover relationships between datasets as metadata – and how can we avoid inadvertently disclosing confidential information in doing so? How do we store metadata like normal data, and can we eventually discard any of it? While such a system faces many challenges before implementation, it is almost necessary as the scale of data and data analysis rapidly increases [10].

VIII. Conclusion

In this paper, existing big data analytics systems and strategies were examined. First, a detailed overview of existing literature in the area was presented. Next, a taxonomy was proposed, dividing the area into four motivations: reducing costs associated with data transfer, efficiently and effectively allocating resources, meeting usability standards and user expectations, and managing the massive data required to justify these systems. Each of these subareas was in turn expounded upon and explored. Finally, some of the open problems in the area were presented.

IX. References

[1] LeFevre, J., Liu, R., Inigo, C., Paz, L., Ma, E., Castellanos, M., & Hsu, M. (2016, June). Building the enterprise fabric for big data with vertica and spark integration. In Proceedings of the 2016 International Conference on Management of Data (pp. 63-75). ACM.

[2] Ananthanarayanan, G., Douglas, C., Ramakrishnan, R., Rao, S., & Stoica, I. (2012, October). True elasticity in multi-tenant data-intensive compute clusters. In Proceedings of the Third ACM Symposium on Cloud Computing (p. 24). ACM.

[3] Vulimiri, A., Curino, C., Godfrey, P. B., Jungblut, T., Padhye, J., & Varghese, G. (2015, May). Global Analytics in the Face of Bandwidth and Regulatory Constraints. In NSDI (Vol. 7, No. 7.2, pp. 7-8).

[4] Chowdhury, M., Kandula, S., & Stoica, I. (2013, August). Leveraging endpoint flexibility in data-intensive clusters. In ACM SIGCOMM Computer Communication Review (Vol. 43, No. 4, pp. 231-242). ACM.

[5] Xuan, P., Ligon, W. B., Srimani, P. K., Ge, R., & Luo, F. (2017). Accelerating big data analytics on HPC clusters using two-level storage. Parallel Computing, 61, 18-34.

[6] Venkataraman, S., Yang, Z., Franklin, M., Recht, B., & Stoica, I. (2016, May). Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (pp. 363-378). USENIX.

[7] Ousterhout, K., Rasti, R., Ratnasamy, S., Shenker, S., & Chun, B. (May 2015). Making Sense of Performance in Data Analytics Frameworks. In Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (pp. 293-307). USENIX.

[8] Costa, P., Donnelly, A., Rowstron, A., & O'Shea, G. (2012). Camdoop: Exploiting In-network Aggregation for Big Data Applications. In Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementations (p. 3). USENIX.

[9] Lin, J., Chen, C., & Chang, J. M. (2013). QoS-Aware Data Replication for Data-Intensive Applications in Cloud Computing Systems. In IEEE Transactions on Cloud Computing (Vol. 1, No. 1, pp. 101-115). IEEE.

[10] Blanas, S., & Byna, S. (2015, March). Towards Exascale Scientific Metadata Management. arXiv:1503.08482v1 [cs.DB].

[11] Nothaft, F., Massie, M., Danford, T., Zhang, Z., Laserson, U., Yeksigian, C., Kottalam, J., Ahuja, A., Hammerbacher, J., Linderman, M., Franklin, M., Joseph, A., & Patterson, D. (2015). Rethinking Data-Intensive Science Using Scalable Analytics Systems. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15).

[12] Gray, J., Liu, D., Nieto-Santisteban, M., Szalay, A., DeWitt, D., & Heber, G. (2005, December). Scientific data management in the coming decade.

[13] Nicolae, B., Costa, C., Misale, C., Katrinis, K., & Park, Y. (June 2017). Leveraging Adaptive I/O to Optimize Collective Data Shuffling Patterns for Big Data Analytics. IEEE Trans. Parallel Distrib. Syst. 28, 6

[14] R. Zhang, H. Wang, R. Tewari, G. Schmidt & D. Kakrania, "Big Data for Medical Image Analysis: A Performance Study," (2016). IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Chicago, IL, pp. 1660-1664.

[15] Melnik, S., Gubarev, A., Long, J., Romer, G., Shivakumar, S., Tolton, M., & Vassilakis, T. (2011, June). Dremel: interactive analysis of web-scale datasets. Commun. ACM 54, 6, 114-123.

[16] Leyshock, P., Maier, D., & Tufte, K. (2014). Minimizing Data Movement through Query Transformation. In 2014 IEEE International Conference on Big Data (pp. 311-316).

[17] Islam, N., Wasi-Ur-Rahman, M., Lu, X., & Panda, D. (2016) High Performance Design for HDFS with Byte-Addressability of NVM and RDMA. In Proceedings of the 2016 International Conference on Supercomputing - ICS 16, 2016.

[18] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., & Vahdat, A. (2010). Hedera: Dynamic Flow Scheduling for Data Center Networks. In Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementations.

[19] Hua, Y., Jiang, H., & Feng, D. (2014). FAST: Near Real-Time Searchable Data Analytics for the Cloud. In SC14:

International Conference for High Performance Computing, Networking, Storage and Analysis.

[20] Ananthanarayanan, G., Hung, M., Ren, X., Stoica, I., Wierman, A., & Yu, M. (2014). GRASS: Trimming Stragglers in Approximation Analytics. In Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14). (pp. 289-302).

[21] SAS. (2018) Big Data Analytics. https://www.sas.com/en_us/insights/analytics/big-data-analytics.html

[22] Three Level Database Architecture. (Aug. 2013) jcsites.juniata.edu/faculty/rhodes/dbms/dbarch.htm.