Toward an Ecosystem for Precision Sharing of Segmented Big Data

Mark Shtern, Bradley Simmons, Michael Smit, and Marin Litoiu
York University, Canada
{mshtern,bsimmons,msmit,mlitoiu}@yorku.ca

***Abstract*—As the amount of data created and stored by organizations continues to increase, attention is turning to extracting knowledge from that raw data, including making some data available outside of the organization to enable crowd analytics. The adoption of the MapReduce paradigm has made processing Big Data more accessible, but is still limited to data that is currently available, often only within an organization. Fine-grained control over what information is shared outside an organization is difficult to achieve with Big Data, particularly in the MapReduce model. We introduce a novel approach to sharing that enables fine-grained control over what data is shared. Users submit analytics tasks that run on infrastructure near the actual data, reducing network bottlenecks. Organizations allow access to a logical version of their data created at runtime by filtering and transforming the actual data without creating storage-intensive stale copies, and resellers can further segment or augment this data to provide added value to analytics tasks. A loosely-coupled ecosystem driven by web services allows for discovery and sharing with a flexible, secure environment that limits the knowledge those running analytics need to have about the actual provider of the data. We describe a proof-of-concept implementation of the various components required to realize this ecosystem, and present a set of experiments to demonstrate feasibility, showing advantageous performance versus storage trade-offs.**

***Keywords*—access control, big data, cloud, hadoop, MapReduce**

I. Introduction

We are living through the most rapid acceleration of data generation in history: 90% of the world's data has only come into existence since 2010[1]. Buried within the vast and ever-expanding store of data is valuable information. This value cuts across disparate domains including: the biological and life sciences [1], [2], where cures for disease are being unravelled from the immense quantities of gathered genomic data; the physical sciences [3], where our understanding of reality is being pieced together at research centers worldwide (e.g., CERN); and the business domain, where companies like Google, Amazon and Facebook seek to monetize every byte of user data to which they gain access.

The more people who have access to this data, the more thoroughly it can be explored and hence the more value can be derived from it; accessibility is one of five key attributes of data described by [4]. While this is apparent to many Internet companies (e.g., Yahoo has made their indices available to the public through Yahoo! Boss[2]) it is only beginning to permeate into the general public's consciousness. For example, the City of Toronto has made much of the data it collects available online[3]. In this work we focus on data accessibility and sharing through the creation of an ecosystem.

MapReduce [5] is a typical method for extracting useful information from large data sets. An open source implementation of MapReduce called Hadoop[4] has become the de-facto standard for Big Data processing. While there have been many recent improvements in the security posture of Hadoop, more work remains to be done and there is ongoing work on this topic [6], [7].

In this paper we introduce the *DaasPatcher ecosystem* which is realized as a marketplace for Big Data sharing on the cloud. This ecosystem enables and facilitates an enhanced data-as-a-service (eDaaS). In an eDaaS, a provider offers data, and the consumer consumes this data by providing code that runs on a provided infrastructure that is local to the data. This provides the consumer with seamless, online access to data they would not otherwise have access to, without requiring the provider to produce stale copies of data and send them over networks not yet ready for Big Data scale data transport.

Each provider in the DaasPatcher ecosystem may determine what data they are willing to share with various types of clients. Clients are differentiated according to various attributes that they possess. The data offerings are advertised within the marketplace. Each defined data offering is generated at runtime by the provider running its internal Map (referred to as a Modifying Map) on the Big Data, passing the results transparently to a customer's MapReduce program. This allows enforcement of the provider's access control policy without additional storage requirements, but also allows the sale and distribution of *segments* of the data; for example, providing access to data from certain years, certain sources, or certain users without actually creating copies of the data. This approach encourages client adoption and participation as it simply requires them to work within the standard MapReduce paradigm. A further benefit of this approach is that data is decoupled from the view that is provided to the client. This affords the provider complete freedom with regard to how and what data is stored / presented (i.e., dynamic constraints can be applied on the fly).

The major technical contributions of this paper are as follows: we propose an approach to facilitate data sharing that build upon Hadoop that that offers benefits in four main areas: the protection of private or confidential information, the segmentation of a large data set based on various dimensions of the data, the ability to abstract the format of the data shared from the underlying data representations, and a novel process referred to as *chaining*. This approach implements a form of data sharing (i.e., need-to-share) in which the data provider is not required to have knowledge about who the data consumer

[1] http://www.viawest.com/sites/default/files/asset/document/ViaWest_IT_Infirmity_Infographic.pdf

[2] developer.yahoo.com/boss/

[3] http://toronto.ca/open

[4] http://hadoop.apache.org

will be. A prototype was built and experiments were run that demonstrate the benefits of this approach.

The remainder of the paper is organized as follows. Section II describes our approach in more detail, introducing the roles and responsibilities of each participant and the overall ecosystem. Section III describes a proof-of-concept implementation and a set of feasibility experiments demonstrating our approach in action, accepting MapReduce jobs to be run on a Hadoop cluster. Section IV discusses extension opportunities and future work. We describe the related work in Section V before concluding the paper (Section VI).

II. Overview

Our approach allows *data analysts* to run MapReduce (MR) jobs on some portion of a *data provider*'s Big Data, while affording the data provider total, fine-grained control over access to each piece of data, and allowing run-time transformation of the data. This run-time mediation is provided by prefixing the user's MapReduce job with an additional Map step (a MapMapReduce, or MMR, job) where the provider can implement access control, data segmentation, and/or data transformation. This *Modifying Map* can also control Map tasks at a low-level, including measuring or limiting execution time. The data analyst is one interested in performing analytics tasks on large-scale data in the cloud, a task of significant interest [8], [9], perhaps for providing services to end users [10].

A data provider may choose to delegate the actual processing to an *infrastructure provider*, which stores a copy of the data on an infrastructure separate from the data provider's production environment, but updates with sufficient frequency to be considered perpetually current. For example, Twitter has delegated most public access of all Tweets to three certified providers[5]. A data provider may also keep such a copy in-house, or it may be acceptable to allow outside access to a single copy of the data. We'll use the term *provider* to refer to an entity responsible for accepting and running MapReduce jobs, regardless of the actual ownership of the data.

The data provider begins by assessing their Big Data, and which portions of it are to be made available outside their organization[6]. The size and scope of the portions can be determined entirely *a priori*, or can be determined only at run-time based on information provided by the user. These logical partitions of the physical data are called *data sources*. There are three primary motivations for establishing restricted subsets of the data for access:

1) **Access control:** Some information within the provider may be appropriate to share with most users, other information may be shared only with one type of user, and the remaining information may never be appropriate to share. For example, Twitter might be willing to provide access to analyze Tweets, but only public ones, and might include some user data with each Tweet, but not physical, email, or IP addresses. This decision could be made *a priori*. Twitter may allow access to more information if the data analyst provides valid credentials giving them access to some private Tweets, a decision that could be made at run-time.
2) **Segmentation:** Not all users need or even desire access to all available information, and not all users can afford access to a complete data set. The data provider can provide useful and marketable subsets of the data. For example, Twitter might provide a segment including only Tweets from Europe or only Tweets from a given month. A provider sharing stock trade data with millisecond accuracy might provide segments per exchange, or per market sector, or per year. This would provide access to Big Data at affordable rates to data analysts unable to afford the complete dataset, or the infrastructure to store/process even a segment of the dataset. These segments could be defined *a priori*. A segment could be defined at run-time to support custom segments, or to allow pay-as-you go (i.e. access is allowed only until pre-purchased credits are consumed).
3) **Transformation / abstraction:** Related to access control, in some cases the provider may wish to share only a transformed version of their data – perhaps de-identified for privacy reasons, or changed to a different data structure. For example, a data provider may not wish to reveal a proprietary compact binary representation of data, and provide instead a JSON-encoded string. Transformations are defined *a priori*, but can be applied selectively at run-time – for example, searching text strings for patterns that resemble phone numbers and obscuring the numbers.

The provider defines their data sources and implements a Modifying Map for each, and provides information about all available data sources (including what data is provided, which Interface the user's Map must implement including the format and data type of the incoming data, the approximate size of the data, etc.) through a web service API. All user interaction happens through this API, and never directly through Hadoop or HDFS. The web service is specified to be the same across all providers, allowing easy integration.

The data analyst implements the prescribed Interface, and submits their compiled code to the provider's web service along with any required parameters. They monitor the status of their job or retrieve the results through the same web service. The user can run their own client for communicating with the web service, or use a client offered through a Software-as-a-Service (SaaS) delivery model, where they submit and monitor jobs through a user interface with the actual communication handled behind-the-scenes. These two scenarios are shown as solid (green) and dashed (red) lines in Fig. 1.

The provider packages the submitted code as a JAR file with their Modifying Map and other supporting code, and executes the MMR job. They respond to requests for progress by querying the Hadoop JobTracker and returning a response. They respond to requests for results by verifying successful job completion, then streaming the results from HDFS.

A. Resellers

This basic user-provider interaction is augmented with *resellers*. While providers have the ability to offer unlimited segmentation and transformation of data, they may prefer to focus on their core competencies, using this approach only to provide access control and privacy protection to the data they are sharing. Resellers may establish relationships with providers and sell access to the provider's infrastructure,

[5]https://dev.twitter.com/programs/twitter-certified-products/products

[6]Our approach can also be employed within an organization, in which case this text might read "available outside business units with complete access"

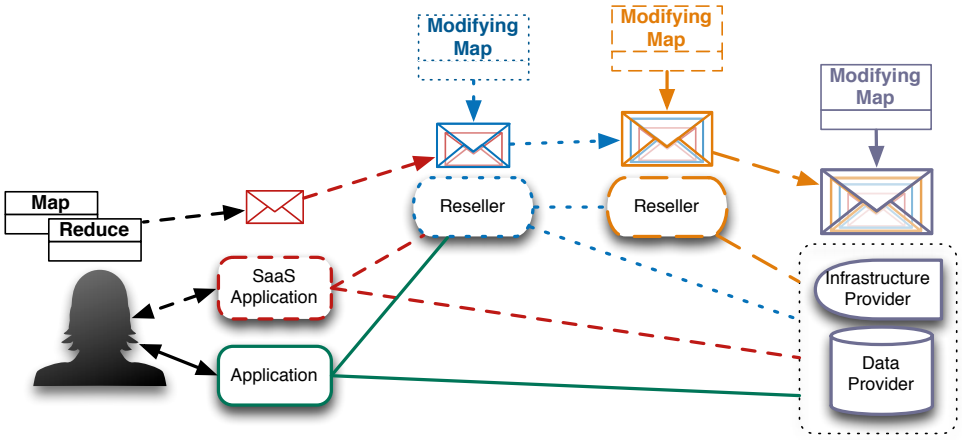


Fig. 1: Various paths along which a MapReduce job can be submitted to a provider. Top envelopes depict resellers wrapping the submitted MapReduce job with an additional Modifying Map which filters data before it reaches the submitted MapReduce job.

accepting MR jobs from users and running them on the provider. A reseller could offer additional segmentation or transformation to produce value-added data sets, or smaller, more affordable data sets. In the Twitter example, one reseller might segment Tweets by estimated household income based on geographic information; another might augment Tweets with a popularity metric; a third might sell subsets of the overall data set where only Tweets mentioning politics or certain products are included. An data analyst could choose one of these smaller data sets to reduce costs. Resellers can be chained together in (theoretically) unlimited series; for example, a fourth reseller might sell segmented access to the first reseller's Tweet+Income data set, by income tax bracket. Fig. 1 shows four paths involving one or two resellers in the dotted (blue) and long-dashed (orange) lines.

To achieve this second (and third, and nth) layer of runtime data mediation, resellers add their own Modifying Maps between the provider's Modifying Map and the user's Map. Because the data provider is the sole arbiter of which data is passed to the reseller, and the reseller then decides which data is sent to the user, each participant retains the control they need. Fig. 2 shows a sequence diagram illustrating the sequence of calls; if any provider or reseller determines the consumer of their service should not have access to a given data record, the Map code is never invoked. Each `map()` invocation may transform the data from the original key-value pair provided to the provider's map method.

A reseller offers the same API as all the providers, allowing users to move among resellers and providers freely. Incoming compiled Map code is augmented with the reseller's Modifying Map, then passed to the next reseller in the chain (or the provider) via their API. Requests for status updates or results are similarly passed on, and the result returned to the requester.

Enhanced Resellers: A reseller adds value as an intermediary by further segmenting or augmenting data from the provider. In some cases, the reseller may host their own infrastructure, acquire data from multiple providers, and run MR jobs directly on their infrastructure where this data is aggregated, filtered, or otherwise combined and transformed. For example, a reseller might offer a data set of users and social trust scores, with data from multiple providers collected

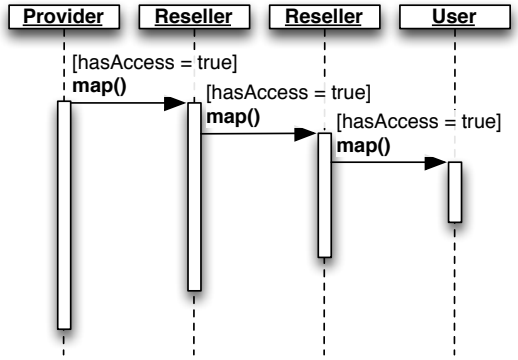


Fig. 2: Within the running MapReduce job, starting from the provider, each participant's map task checks to see if the next participant has access to a given data record before invoking the next `map(key, value, out, reporter)` method.

into a local Hadoop instance that accepts MR jobs from users. In this case they appear to the data analyst as the provider, and function as a provider when receiving jobs for their own data sources. When they run jobs to acquire copies of the data from providers, they behave like data analysts. They may also function as a normal reseller, accepting MR jobs for submission to the provider. Managing these multiple roles is the responsibility of the reseller; throughout this text, when we refer to providers we include Enhanced Resellers when in their provider role, and likewise for reseller and user roles.

B. The DaaSPatcher Ecosystem

Considering only the two previous sections, our approach allows for users having established relationships to a reseller and/or a provider to submit MR jobs to a provider (optionally via a reseller). This is sufficient for public data, where the provider only supplies information that they are willing to made public. In this section, we describe an approach to Attribute-based Access Control (ABAC) [11] which, when combined with the features of the web service offered by each provider/reseller, facilitates a larger ecosystem for sharing public, semi-private, and private data sets with verified users. In the DaaSpatcher Ecosystem, users can discover available

data sources and submit jobs to them easily, and providers can authorize users to run MR jobs without knowing all of the details about the user or having an established relationship.

In our approach to ABAC, a user registers with a central service (for convenience, we call this the *marketplace*, a distributed, reliable service). They can add attributes to their account by simply adding them (user-signed attributes) or by requesting that a third-party *authority* recognized by the marketplace provide validated attributes. Potential authorities include Facebook, Twitter, or Google accounts (through OpenID); Verisign or PKI trust establishment regimes; companies that hold records on individuals like Equifax; or other organizations. Each authority can assign the user an attribute in the authority's namespace, and sign it with their key. They can optionally include metadata with each attribute specifying their level of confidence in the accuracy of the provided attribute.

When a provider publishes information about a data source to the marketplace they include two sets of attributes. The first attribute set is used to specify what attributes must be submitted in order for the provider to verify access to a particular data source. The provider would determine this set based on their level of trust for each authority's attributes; for example, some might find the presence of a Facebook account sufficient proof that the user is over the age of thirteen, while others would require additional evidence. The user – or the reseller on their behalf – includes these required attributes in their request, and the provider compares the value of the attributes to their requirements. This requires established trust agreements among the resellers and involved providers; a small-scale solution would be off-line informal trust agreements; at a larger scale, a framework for establishing or negotiating trust can be employed (e.g., [12]). A provider may register as an authority and require attributes only they can assign, which would allow them to control the mechanisms for authorizing users more completely. The second attribute set is used to specify what attributes a user must have in order to view metadata about a particular data source. The marketplace is responsible for enforcing this limit.

Users are informed of the attributes required, and the resellers involved in the chain, for any data source to which they wish to submit MapReduce jobs. They have the opportunity to acquire the additional attributes from authorities if required, or to choose an alternative data source. They will not see the provider's rules; simply having the required attributes is not sufficient to run a MR job. For example, one attribute might be 'age' as verified by a credit card company; a user sends this signed attribute to the provider, and the provider checks the value against its rules to assess whether the user is authorized to run the submitted MR job on its data source.

For convenience, the marketplace maintains a list of available attributes from all registered authorities. Providers can specify a given attribute (e.g. age) from a specific authority, or from any authority verifying that attribute. The marketplace also maintains quality/satisfaction ratings of each provider and reseller, which users can use to identify which resellers and providers they might be willing to send their attributes through. It may hide certain data sources from users based on its own rules (e.g. depending on what package the user purchased) or on constraints expressed by the provider. The marketplace may provide a web interface with which data analysts interact with data providers and authorities; it may also build on existing work in the services community regarding automated service discovery [13].

III. Implementation & Experiments

We authored a proof-of-concept implementation implementing key elements of our approach, particularly the ability for a provider to run user code in Hadoop while asserting control on what the user had access to. The implementation details of this service-oriented system are available in [14]. We conducted a series of experiments to assess the behavior and performance of our implementation to demonstrate the feasibility of our approach.

A. Implementation

Each defined data source must have an associated Modifying Map that is responsible for both filtering and transforming the data it receives. The design pattern for this Modifying Map is based on the Delegation pattern, and is central to our approach. Recall that Hadoop reads the input data and calls the `map()` method of the MapReduce job with data records. The `map()` method of a ModifyingMap a) invokes a `filter()` method that decides if this data source will provide access to that data element or not, and if so executes any transformations on the data; and b) delegates the processing of any data that comes through the filter to the user- or reseller-submitted Map class. To achieve the delegation, the submitted classes (which are in a known location, see below) are loaded via Java Reflection and their `map()` method is invoked. This process repeats, with the `map()` method of all ModifyingMaps being called in order until either a) any Modifying Map decides to not provide access to the given data element, or b) the data analyst's `map()` method is reached, and the actual analytics task is performed. A reference implementation of a Modifying Map pattern is supplied to resellers and providers.

We implemented the Modifying Maps required to produce three data sources: A (a provider), B (a reseller), and C (a second reseller). The A Modifying Map filters out all words from the input text except those beginning with the letter 'a'. The B Modifying Map modifies A, passing on only words beginning with the letters 'ab'. The C Modifying Map also modifies A, but permits words beginning with either 'ac' or 'c'. They should not have access to any data values beginning with 'c'; this is an attempt to gain illicit access to information.

We implemented RESTful web services for providers and resellers, and a RESTful web service client in the form of a web application offered as a service to users. The provider web service receives submitted JAR files, and places the submitted code in a predefined location in its own JAR file. Constraining requesters to use a specified package name as the base for all of their code prevents users from supplanting the provider's code or standard libraries. It then uses the Hadoop client to submit jobs to a remote Hadoop cluster. It is backed by a standard relational database which maintains metadata about the data source (the ModifyingMap and Filter with all supporting code, what user attributes are required, the location of the source data set in HDFS) and the submitted jobs (unique IDs, the identity of the submitter, the HDFS location of the results, the Hadoop job ID). Status requests return one of rejected,

pending, executing, or completed. Also supported is a tracker request, which returns a modified version of the HTML page produced by the JobTracker for the named Hadoop job. To answer requests for results, the service reads from the remote HDFS drive and streams them to the user (a small buffer is used to avoid loading the entire result file into memory).

The reseller web service also adds the submitted binary code to its own JAR file corresponding to the given job ID, then consults a backing relational database to find out to which provider (or reseller) this data source corresponds. It invokes the API of the next participant in the chain. Requests for status, trackers, or results are handled by similar lookups, the request of the response from the next participant, the receipt of that response, and the return to the original requester. Though this introduces overhead at each level of redirection, such requests are infrequent and do not impact the MR job.

The marketplace is a software-as-a-service RESTful client that offers a web interface. Users register, and provide the attributes of email address (verified via a confirmation email) and resident country (self-validated). The marketplace associates permission to access various data sources with groups (i.e. the subscription package), and users are assigned to groups based on their attributes (with administrator intervention). Users see a list of data sources to which they have access, and may submit jobs, track job status, view job trackers, and download job results. The focus of the implementation was to fully explore data segmentation, data transformation, and chaining; some elements of our approach to ABAC such as provider-side verification of user attributes and the distributed authority infrastructure are not yet implemented.

To represent the data analyst, we created a basic Word-Count MR job (based on standard Hadoop examples) that can be packaged in a jar file and submitted to a reseller or provider.

B. Experiment Design

In the first experiment, we submitted the Wordcount MapReduce job through the reseller and provider APIs and examined the results returned by the API. Though we tested various combinations, of particular interest is the reseller (C) that attempted to access data outside of what the provider was offering (A). We expected that the results produced by the reseller (C) will still be a strict subset of the provider (A).

In the second experiment, we examined performance. After running the Wordcount MapReduce job on the provider directly, we submitted the same map-reduce job via the reseller and provider APIs that are providing access to data filtered by Modifying Maps A, B, and C. We repeated each MapReduce job 5 times, then compared the average processing time for each type of MapReduce job. We used the JobTracker web user interface to track job progress on the Hadoop cluster and record processing time. We hypothesized that performance would not degrade; some filtering overhead is to be expected, and the entire data file must be read for all scenarios, but the reduce phase should proceed more quickly.

In the third experiment, we examined the separation of the participating entities. Fictional users (Alice, Bob, and Charlie) each submitted a MapReduce task to (respectively) the provider's API (A), and two reseller APIs (B and C). We monitored the jobs from the standpoint of all three users, the reseller, and provider, in particular what information about the running jobs was available to each party. We expected each user would see only their jobs, and would not be able to infer meaningful information from the results provided.

C. Experiment Environment

We used Cloudera Manager (free edition) to install Hadoop on nine Amazon EC2 `m1.medium` instances[7] running Ubuntu 12.04.1 LTS. The version of Hadoop installed is Cloudera's build, CDH 4.1.3, with the `hdfs`, `mapreduce`, and `zookeeper` services configured with all default Cloudera settings. Cloudera Manager chose one node to be the NameNode, JobTracker, and Zookeeper server; all nodes (including the master) participated as both data nodes and task nodes. One slot was configured per core (and therefore per node). The resulting HDFS virtual drive was 3.37 TB, with a NameNode max heap size of approximately 200 MB. Cloudera Manager by default selects the FairScheduler, which allows multiple jobs to run simultaneously, rather than the FIFO scheduler that is the Hadoop default.

We used a dataset consisting of Apache Software Foundation email archives from the last decade [15]. After removing much of the non-text data and aggregating the per-month files into larger files, the dataset was 42 GB in 105 files. This sizing was chosen to demonstrate our approach on a relatively large dataset while still allowing for multiple iterations of all experiments within a reasonable time; the ability of Hadoop to process many terabytes of data is not in question in these experiments. The total data processed across all experiments exceeded 1 TB.

Our user, reseller, and provider API implementations were deployed to web servers, the user code to Rackspace (512 MB[8]) and the reseller and provider to Amazon EC2 (t1.micro[9]). The provider web server was connected to the Hadoop cluster and authorized to read a specific HDFS directory and launch MapReduce jobs.

D. Results

In all of the experiments, 385 map tasks and a single reduce task were created. All of the map tasks ran on node-local data.

After performing the *first experiment*, we both manually and programmatically examined the results to verify only the data available to each data source was present in the results file. There were no words in the results that should not have been in the input data. In particular, although Reseller C strove to provide words beginning with 'ac' and 'c', it returned exclusively words beginning with 'ac', as it only had access

[7] An m1.medium instance is approximately 80% of a Intel(R) Xeon(R) E5507 @ 2.27GHz processor, 3.7GB of RAM, with 400 GB of instance storage, and an on-demand price of 0.130 per hour for Linux instances in the US East region.

[8] The (first generation) 512 MB instance is named after the amount of RAM available. Each is guaranteed 3.1% of each of four virtual CPUs (backed by quad-core AMD Opteron 2GHz processors), and can burst up to 100% of each core when capacity is available. They are provisioned for 40Mbps network connections and with 20 GB of storage.

[9] A t1.micro instance offers short periodic bursts up to a single core Xeon processor, no sustained CPU usage, and 613 MB RAM.

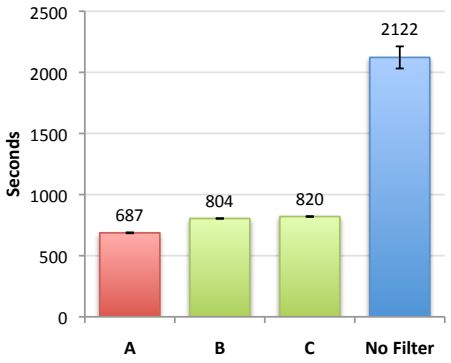


Fig. 3: The average time (across five runs) to complete a MapReduce job comparing a normal MR job to using Modifying Maps.

to data that had already passed the provider's Modifying Map A which removed all words other than those starting with 'a'.

The average processing times (and standard deviations) from the multiple runs conducted in the *second experiment* are shown in Fig. 3. The filtered/transformed MapReduces surpassed our expectations for performance, consistently completing in approximately one-third the time of the unfiltered MapReduce: 11-13 minutes versus over 35 minutes. We expected the reading of the information from HDFS to be a larger factor in overall processing time.

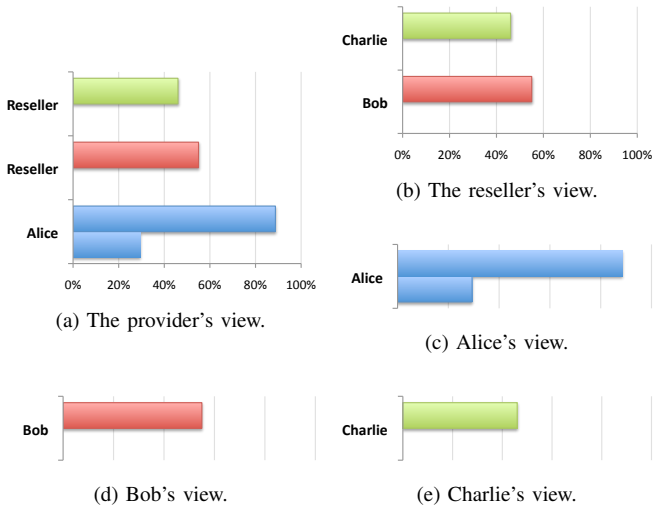
Transferring the results to the end user was not included in the time calculations, though we do note that transmitting the result of the MapReduce would be substantially faster than transmitting the complete data set: the jobs sent produced results that ranged in size from 4 MB (B) to 200 MB (A), compared to 42,000 MB for the raw data. Similarly, a post-processing approach modifying the results of the reduce would still be left with substantial processing: the unfiltered MapReduce produced a 5,000 MB results file.

In the *third experiment*, we assessed the separation of the various roles in our implementation. Fig. 4 shows a visualization of what each participant viewed: each user was aware only of their job and the site to which they submitted it, the reseller was aware only of the two jobs submitted to them, and the provider was aware of all three jobs but did not know about the original submitter.

We examined the CPU usage of the reseller's web host and the provider's Hadoop cluster during this experiment (Fig. 5). We verified that other than some low CPU usage at the beginning of the job as it was launched and the users checked the job status more frequently, the reseller machine was idle while the Hadoop cluster averaged 100% CPU utilization for over 30 minutes. This demonstrates that these jobs were effectively transferred to the data provider as designed.

E. Discussion

Another possible approach to segmented data sharing would be to create multiple copies of the data in the segments or with the transformations required, and provide access directly to these subsets. This would remove the need to re-read the entire data set for an MR job that will process only



(a) The provider's view.

(b) The reseller's view.

(c) Alice's view.

(d) Bob's view.

(e) Charlie's view.

Fig. 4: The jobs visible to each participant in the data sharing environment when multiple jobs are running; shown is the map/reduce progress for each job after approximately 30 minutes.

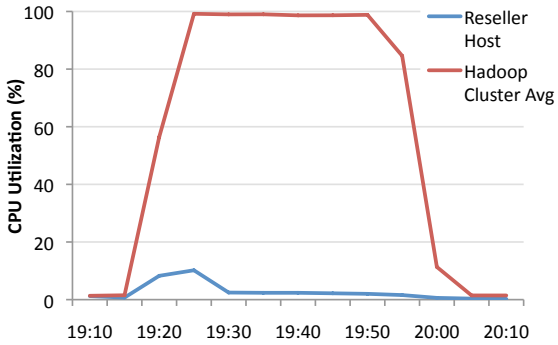


Fig. 5: CPU utilization of the Hadoop cluster and the reseller during the multi-job scenario.

a subset. However, this would increase storage overhead (a significant concern with Big Data), would have the potential for a rapidly growing number of copies, and would introduce consistency/staleness problems keeping all copies up to date. Additionally, this would effectively require resellers to maintain their own copies of the data sources they offer, which would reduce the reward a reseller receives for participating in the system by adding value.

With regards to the second experiment we note the significant improvement in time to completion (i.e., approximately one third the time to completion) of the Hadoop jobs using the DaaSPatcher ecosystem. We infer that this speedup is related to the fact that the Modifying Map is acting as a filter and hence is reducing the total amount of processing required for the various jobs, offsetting any incidental overhead. This implies that Modifying Maps may also be used to help a provider improve the throughput of their Hadoop cluster.

For the third experiment the primary statistic available to each participant was the progress of the map and reduce tasks (this statistic is visualized in Fig. 4 as measured at a point in time 22 minutes into the experiment), but through the

tracker feature they could also access the counters provided by Hadoop. By measuring the number of simultaneous map tasks executing, the user could infer the relative load on the server – but only if they knew the exact size and configuration of the cluster, which by design of this ecosystem would typically not be the case. However, by assessing the number of map tasks that ran data-local and comparing that to the number of tasks executing simultaneously, information could be gleaned about the capacity of the cluster. This information is no longer included in the tracker.

It is worth noting that the Hadoop cluster used Amazon spot instances, which allows users to bid on unused capacity, and pay only the prevailing market price. We paid $.013/hr for each instance, or a total cost of $1.17 for the 10 hours of processing time required while these experiments ran. Had this been run using on-demand instance it would have resulted in a cost an order of magnitude higher (e.g., $11.70).

IV. Observations and Thoughts Going Forward

Attribute Management: This represents a known challenge when dealing with ABAC. While there has been work done that addresses this issue [11], [16] in our approach we require attributes be specified under authority namespaces in order to prevent attribute collisions.

Client Development Environment: This refers to the development environment of the client to be utilized during the development phase of their MapReduce program. There are two possible approaches to building this environment:

1) A small (i.e., 1 GB) dataset could be supplied to the clients local system for them to work on. A problem with this approach is that once data is released it can not be recovered. A second problem is that the developer must have a working version of Hadoop deployed.
2) The provider / reseller could provide a working development environment to the client that includes an installed Hadoop/data as a virtual machine image which they could run as an instance and upon which they could practice. A problem with this approach is designing a bundle that can be used only for a limited period of time. This represents a possible avenue of future work.

Verification of Client Maps: A potential security concern exists in the running of untrusted client code on a provider's system. Future work will explore the potential for running the submitted code in a sandboxed environment prior to executing on the live system. Additionally, Hadoop 2x[10]has introduced several security [6] that will also help to mitigate this concern.

Verification of Modifying Map: While the current work did not focus on the verification of Modifying Maps this is a crucial aspect with regards to the integrity of the DaaS-Patcher ecosystem. In the current model, the provider/reseller is completely responsible to verify the correctness of their Modifying Maps. We intend to explore applying model driven techniques [17] to ensure the correctness of these maps as a part of our future work.

Educational Tool: Installing, deploying and configuring a Hadoop cluster ranges from very complicated to reasonably easy. One of the main aspects that determines this level of complexity is the person's prior experience working with such a system. By removing the need to perform all administrative tasks and by virtue of the simplified interface (i.e., a high-level GUI rather than complex CLI commands) the student's focus can remain solely on the logic of the MapReduce program they are constructing allowing them to develop their skills and understanding immediately. Further, because we can deploy different Modifying Maps to each student, there is no worry about cheating on assignment material, etc.

Elasticity: A key driver of the DaaSPatcher ecosystem design is the fact that it is built on the cloud. The ability of applications to scale dynamically in accordance with their elasticity policy's [18] (i.e., in response to changing demands/workloads) is critical when designing infrastructure for unpredictable numbers of users with undefined requirements.

Hybrid clouds: To enable an organization to extend a Hadoop cluster to the public cloud, we could use a Modifying Map to selectively copy data to the public Hadoop cluster. Other Modifying Maps can be annotated and partitioned (as in [19]), moving those that can run in the public cloud to the new public Hadoop cluster.

V. Related Work

The Clark-Wilson integrity model [20] defines a set of nine (i.e., five certification C1-C5 and four enforcement E1-E4) rules that ensure the integrity of data. Four main elements are defined: a Constrained Data Item (CDI), an Unconstrained Data Item (UDI), Integrity Verification Procedure (IVP) and Transformation Procedure (TP) upon which these rules are specified. Our framework borrows conceptually in many ways from this model . For example, we follow C5 during data entry and publishing phases. Specifically, when data is first uploaded to a providers HDFS can be considered as a UDI. The procedure that facilitates its upload can be seen as a TP and the final data store is seen as a certified CDI. Alternatively, in the case of a reseller publishing a data source from provider this is also the case in that prior to publication the data is seen as a UDI. Then, upon verification of its signature via a TP it is certified as a CDI. Other examples follow a similar pattern.

Previous work [11] has observed the effectiveness of using ABAC on the Internet. The ABAC approach to access control focuses on the attributes of both subjects and objects and uses these to define who may act on what avoiding the need for explicit knowledge of the various entities. Our specified ecosystem takes advantage of ABAC allowing for dynamic Modifying Map selection.

Singhal et al. [21] present a framework that addresses issues of security implicit when collaborating and sharing resources across a multi-cloud. We considered similar issues in the design of the DaaSpatcher ecosystem notably mechanisms for maintaining attribute privacy and for security delegation.

Airavat [7] is a system that assures high degrees of security and privacy for MapReduce problems through its use of mandatory access control (i.e., SELinux) and differential privacy. While we find this work inventive, and we share the same main objective (i.e., privacy/sharing of data) our approach differs in several key ways. First, we do not require

[10]http://hadoop.apache.org/docs/current/

labelling of data. Second, while we limit the possible map functions supplied to any particular individual we do not prevent a user from utilizing any mapper or reducer which they compose. Third, our approach in no way modifies any component of the Hadoop MapReduce framework, filesystem or the Java virtual machine. Finally, we support the standard MapReduce programming model.

Adopters of Hadoop in industry recognize the need to improve the security capabilities of Hadoop (e.g. [6]). A spectrum of solutions and approaches are currently being explored. One interesting approach [22] involves the inclusion of hooks / callbacks at points in MapReduce to facilitate various types of fine and coarse grained control.

With interest in Big Data rapidly expanding [23] it is crucial that comprehensive security solutions be introduced to address things like complex and distributed node management, open communication channels, etc. [24]. The AERIE architecture [25], [26] offers a highly scalable, secure and comprehensive, adaptive, multi-cloud application management framework within which a Big Data stack can be run with little or no modification in a secure fashion.

VI. Conclusion

In this paper, we introduce the DaaSpatcher ecosystem, a methodology for secure, segmented Big Data sharing on the cloud. Modifying Maps are the kernel of our solution and allow the provider to enforce their data access policy and assure data privacy, unencumbered by direct knowledge of their clients, while also seamlessly splitting Big Data into shareable, saleable segments. Resellers are added to augment the model and facilitate complex chains of propagated rights without compromising the integrity of the provider's data sharing policy. An implementation of the main components of this marketplace was developed and results of experiments presented to demonstrate the benefits and functionality of this novel approach to Big Data sharing on the cloud.

Acknowledgment

This research was supported by IBM Centres for Advanced Studies (CAS), the Natural Sciences and Engineering Council of Canada (NSERC) under the Smart Applications on Virtual Infrastructure (SAVI) Research Network, and the Ontario Research Fund under the Connected Vehicles and Smart Transportation partnership.

References

[1] S. Vijayakumar, A. Bhargavi, U. Praseeda, and S. Ahamed, "Optimizing sequence alignment in cloud using hadoop and mpp database," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, june 2012, pp. 819–827.

[2] E. Waltz, "1000 genomes on amazon's cloud," *Nat Biotech*, vol. 30, no. 5, pp. 376–376, 2012.

[3] S. Toor, R. Toebbicke, M. Resines, and S. Holmgren, "Investigating an open source cloud storage infrastructure for CERN-specific data analysis," in *Networking, Architecture and Storage (NAS), 2012 IEEE 7th International Conference on*, 2012, pp. 84–88.

[4] A. Barua, D. Mani, and R. Mukherjee, "Measuring the business impacts of effective data: Chapter one of a three-part study," The University of Texas at Austin Sponsored by Sybase, Tech. Rep., 2010.

[5] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[6] O. O'Malley, K. Zhang, S. Radia, R. Marti, and C. Harrell, "Hadoop security design," Yahoo, Inc., Tech. Rep., 2009.

[7] I. Roy, S. T. Setty, A. Kilzer, V. Shmatikov, and E. Witchel, "Airavat: Security and privacy for MapReduce," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*. USENIX Association, 2010, pp. 20–20.

[8] A. Cuzzocrea, I.-Y. Song, and K. C. Davis, "Analytics over large-scale multidimensional data: the big data revolution!" in *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP*, ser. DOLAP '11. New York, NY, USA: ACM, 2011, pp. 101–104.

[9] I. Konstantinou, E. Angelou, D. Tsoumakos, and N. Koziris, "Distributed indexing of web scale datasets for the cloud," in *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*, ser. MDAC '10. New York, NY, USA: ACM, 2010, pp. 1–6.

[10] H. Vashishtha, M. Smit, and E. Stroulia, "Migrating a legacy web-based document-analysis application to Hadoop and HBase: An experience report," in *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*. IGI Global, 2012, pp. 226–247.

[11] T. Priebe, W. Dobmeier, and N. Kamprath, "Supporting attribute-based access control with ontologies," in *The First International Conference on Availability, Reliability and Security*. IEEE, 2006, pp. 465–472.

[12] C. H. Yew and H. Lutfiyya, "A middleware-based approach to supporting trust-based service selection," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, 2011, pp. 407–414.

[13] M. Rambold, H. Kasinger, F. Lautenbacher, and B. Bauer, "Towards autonomic service discovery," in *Services Computing, IEEE International Conference on*. IEEE Computer Society, 2009, pp. 192–201.

[14] M. Smit, B. Simmons, M. Shtern, and M. Litoiu, "Enabling an enhanced data-as-a-service ecosystem," in *IEEE Congress on Services, Cloud Cup*, 2013, To Appear.

[15] G. Ingersoll, "Apache Software Foundation public mail archives," Available via http://aws.amazon.com/datasets/7791434387204566, original source http://mail-archives.apache.org/mod_mbox/, August 2011.

[16] A. H. Karp, H. Haury, and M. H. Davis, "From ABAC to ZBAC: The evolution of access control models," HP Laboratories, Tech. Rep. HPL-2009-30, 2009.

[17] S. Sendall and W. Kozaczynski, "Model transformation: the heart and soul of model-driven software development," *Software, IEEE*, vol. 20, no. 5, pp. 42 – 45, 2003.

[18] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai, "Exploring alternative approaches to implement an elasticity policy," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011, pp. 716–723.

[19] M. Smit, M. Shtern, B. Simmons, and M. Litoiu, "Partitioning applications for hybrid and federated clouds," in *CASCON '12: Proceedings of the 2012 conference of the Center for Advanced Studies on collaborative research*, 2012, pp. 27–41.

[20] D. D. Clark and D. Wilson, "A comparison of commercial and military computer security policies," in *Proceedings of the 1987 Symposium on Security and Privacy*, April 1987.

[21] M. Singhal, S. Chandrasekhar, T. Ge, R. Sandhu, R. Krishnan, G.-J. Ahn, and E. Bertino, "Collaboration in multicloud computing environments: Framework and security issues," *Computer*, vol. 46, no. 2, pp. 76–84, 2013.

[22] S. Saklikar, "Embedding security and trust primitives within map reduce," http://www.emc-china.com/rsaconference/2012/en/download.php?pdf_file=TC-2003_EN.pdf, 2012.

[23] A. McAfee and E. Brynjolfsson, "Big data: The management revolution," *Harvard business review*, vol. 90, no. 10, pp. 60–128, 2012.

[24] Securosis, "Securing big data: Security recommendations for Hadoop and NoSQL environments," Tech. Rep., 2012, last accessed February 19, 2013 at https://securosis.com/assets/library/reports/SecuringBigData_FINAL.pdf.

[25] M. Shtern, B. Simmons, M. Smit, and M. Litoiu, "An architecture for overlaying private clouds on public providers," in *8th International Conference on Network and Service Management, CNSM 2012, Las Vegas, USA*, 2012, pp. 371–377.

[26] ——, "Navigating the cloud with a MAP," in *13th IFIP/IEEE International Symposium on Integrated Network Management (IM)*. To Appear, 2013.