Crossing Analytics Systems: A Case for Integrated Provenance in Data Lakes

Conference Paper · October 2016

DOI: 10.1109/eScience.2016.7870919

2 authors:

Isuru Suriarachchi
Indiana University Bloomington
11 PUBLICATIONS 118 CITATIONS
SEE PROFILE

Beth Plale
Indiana University Bloomington
275 PUBLICATIONS 5,116 CITATIONS
SEE PROFILE

Crossing Analytics Systems: A Case for Integrated Provenance in Data Lakes [Preprint, eScience 2016]

Isuru Suriarachchi and Beth Plale
School of Informatics and Computing, Indiana University
Bloomington, IN, USA
Email: {isuriara,plale}@iu.edu

***Abstract*—The volumes of data in Big Data, their variety and unstructured nature, have had researchers looking beyond the data warehouse. The data warehouse, among other features, requires mapping data to a schema upon ingest, an approach seen as inflexible for the massive variety of Big Data. The Data Lake is emerging as an alternate solution for storing data of widely divergent types and scales. Designed for high flexibility, the Data Lake follows a schema-on-read philosophy and data transformations are assumed to be performed within the Data Lake. During its lifecycle in a Data Lake, a data product may undergo numerous transformations performed by any number of Big Data processing engines leading to questions of traceability. In this paper we argue that provenance contributes to easier data management and traceability within a Data Lake infrastructure. We discuss the challenges in provenance integration in a Data Lake and propose a reference architecture to overcome the challenges. We evaluate our architecture through a prototype implementation built using our distributed provenance collection tools.**

I. Introduction

Industry, academia, and research alike are grappling with the opportunities that Big Data brings in mining data from numerous sources for insight, decision making, and predictive forecasts. These sources (*e.g.,* clickstream, sensor data, IoT devices, social media, server logs) are frequently both external to an organization and internal. Data from sources such as social media and sensors is generated continuously. Depending on the source, data can be structured, semi-structured, or unstructured. The traditional solution, the data warehouse, is proving inflexible and limited [1] as a data management framework in support of multiple analytics platforms and data of numerous sources and types.

The response to the limits of the data warehouse is the Data Lake [2], [1]. A feature of the Data Lake is its schema-on-read (as opposed to schema-on-write which happens at ingest time) where commitments to a particular schema are deferred to time of use. Schema-on-read suggests that data are ingested in a raw form, then converted to a particular schema as needed to carry out analysis. The Data Lake paradigm acknowledges that high throughput analytics platforms in use today are varied, so has to support multiple Big Data processing frameworks like Apache Hadoop[1], Apache Spark[2] and Apache Storm[3]. The vision of the Data Lake is one of data from numerous sources being dropped into the lake quickly and easily, with tools around the lake as designated fishers of the lake, intent on catching insight by the rich ecosystem of data within the Data Lake.

This greater flexibility of the Data Lake leads to rich collections of data from various sources. It, however, leaves greater manageability burdens in the hands of the lake administrators. The Data Lake can easily become a “dump everything” place due to absence of any enforced schema, sometimes referred to in literature as a “data swamp” [3]. The Data Lake could easily ignore the fact that data items in a Data Lake can exist in different stages of their life cycle. One data item may be in raw stage after recent generation where another may be the fined result of analysis by one or more of the analysis tools. The complications of data life cycle increases the need for proper traceability mechanisms. In this paper the critical focus of our attention is on metadata and lineage information through a data life cycle which are key to good data accessibility and traceability [4].

Data provenance is the information about the activities, entities and people who involved in producing a data product. Data provenance is often represented in one of two standard provenance representations (*i.e.,* OPM [5] or PROV [6]). Data provenance can help with management of a Data Lake by making it clear where an object is in its lifecycle. This information can ease the burden of transformation needed by analysis tools to operate on a dataset. For instance, how does a researcher know what datasets are available in the lake for Apache Spark analysis, and which can be available through a small amount of transformation? This information can be derived by hand by the lake administrator and stored to a registry, but that approach runs counter to the ease with which new data can be added to the data lake. Another issue with the Data Lake is trust. Suppose a Data Lake is set up to organize data for the watershed basin of the Lower Mekong River in southeast asia. The contributors are going to be from numerous countries through which the Mekong River passes. How does the Data Lake ensure that the uses of the data in the lake are proper and adhere to the terms of contribution? How does a researcher, who uses the Data Lake for their research, prove that their research is done in a way that is consistent with the terms of contribution?

Provenance contributes to these questions of use and trust. If the Data Lake framework can ensure that every data product’s

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

[2] http://spark.apache.org/

[3] http://storm.apache.org/

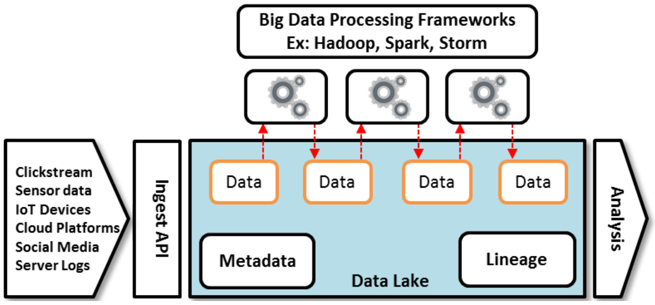


Fig. 1. Data Lake Architecture

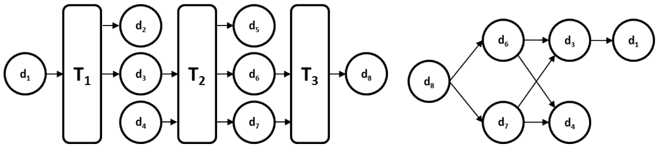
lineage and attribution are in place starting from the origin, critical traceability can be had. However, that is a challenging task because a data product in a Data Lake may go through different analytics systems such as Hadoop, Spark and Storm which do not produce provenance information by default. Even if there are provenance collection techniques for those systems, they may use their own ways of storing provenance or use different standards. Therefore generating integrated provenance traces across systems is difficult.

In this paper we propose a reference architecture for provenance in a Data Lake based on a central provenance subsystem that stores and processes provenance events pumped into it from all connected systems. The reference architecture, which appeared in early work as a poster [7], is deepened here. A prototype implementation of the architecture using our distributed provenance collection tools shows that the proposed technique can be introduced into a Data Lake to capture integrated provenance without introducing much overhead.

The paper's three main contributions are: identification of the data management and traceability problems in a Data Lake that are solvable using provenance highlighting the challenges in capturing integrated provenance. Second, a reference architecture to overcome those challenges. Third, an evaluation of the viability of the proposed architecture using a prototype implementation with techniques that can be used to reduce the overhead of provenance capture.

II. Data Lake Architecture

The general architecture of a Data Lake, shown in Figure 1, contains three main activities: (1) Data ingest, (2) Data processing or transformation and (3) Data analysis. A Data Lake may open up number of ingest APIs to bring data from different sources into the lake. In most cases, raw data is ingested into the Data Lake for later use by researchers for multiple purposes at different points of time. Activity in the Data Lake can be viewed as data transformation: where data in the Data Lake is input to some task, and output is stored back to the Data Lake. Modern large scale distributed Big Data processing frameworks like Hadoop, Spark and Storm are the source of such transformations, especially for Data Lakes implemented on HDFS. Mechanisms like scientific workflow systems such as Kepler [8] and legacy scripts may apply as well. As shown in Figure 1, a data product created as an output from one transformation can be an input into another transformation which itself may produce another one as a result. Finally when all processing steps are done, resulting data products are used for different kinds of analysis reports and predictions.



(a) Data Flow (b) Data Lineage

Fig. 2. Data Lineage in a Data Lake

III. Provenance in Data Lake

A. Role of Provenance

The Data Lake achieves increased flexibility at the cost of reduced manageability. In the research data environment, when differently structured data is ingested by different organizations through one of multiple APIs, tracking becomes an issue. Chained transformations that continuously derive new data from existing data in the lake further complicate the management. How can minimal management be added to the lake without invalidating the attractive benefit of ease of ingest? We posit that this minimal management is in the form of mechanisms to track origins of the data products, rights of use and suitability of transformations applied to them, and quality of data generated by the transformations. Carefully captured provenance can satisfy these needs allowing, for instance, answers to the following two questions: 1.) Suppose sensitive data are deposited into a Data Lake; social science survey data for instance. Can data provenance prevent improper leakage into derived data? 2.) Repeating a Big Data transformation in a Data Lake is expensive due to high resource and time consumption. Can live streaming provenance from experiments identify problems early in their execution?

B. Challenges in Provenance Capture

Data in a Data Lake may go through number of transformations performed using different frameworks selected according to the type of data and application. For example, in a HDFS based Data Lake, it is common to use Storm or Spark Streaming for streaming data and Hadoop MapReduce or Spark for batch data. Other legacy systems and scripts may be included as well. To achieve traceability across transformations, provenance captured from these systems must be integrated, a challenge since many do not support provenance by default.

Techniques exist to collect provenance from Big Data processing frameworks like Hadoop and Spark [9], [10], [11], [12]. But most are coupled to a particular framework. If the provenance collection within a Data Lake depends on such system specific methods, provenance from all subsystems should be stitched together to create a deeper provenance trace. There are stitching techniques [13], [14] which bring all provenance traces into a common model and then integrate them together. However the process of converting provenance

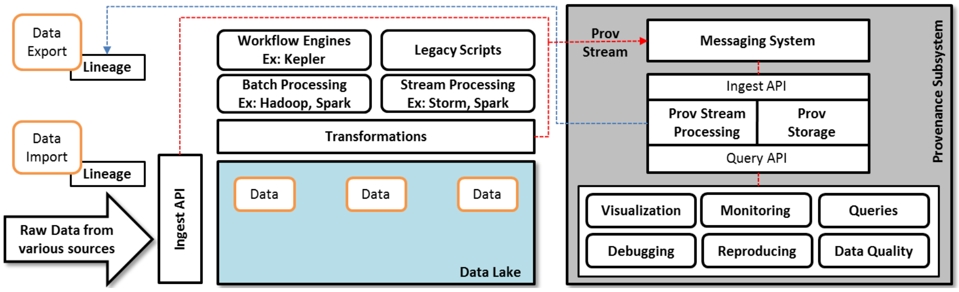


Fig. 3. Provenance for Data Lakes: Reference Architecture

traces from different standards into a common model may lose provenance information depending on the data model followed by each standard. As a Data Lake deals with Big Data, most transformations generate large provenance graphs. Converting such large provenance graphs into a common model and stitching them together can introduce considerable compute overheads as well.

C. Provenance Integration Across Systems

To address provenance integration, we propose a central provenance collection system to which all components within the Data Lake stream provenance events. Well accepted provenance standards like W3C PROV [6] and OPM [5] represent provenance as a directed acyclic graph ($G = (V, E)$). A node ($v \in V$) can be an activity, entity or agent while an edge ($e = \langle v_i, v_j \rangle$ where $e \in E$ and $v_i, v_j \in V$) represents a relationship between two nodes. In our provenance collection model, a provenance event always represents an edge in the provenance graph. For example, if process p generates the data product d, the provenance event adds a new edge ($e = \langle p, d \rangle$ where $p, d \in V$) into the provenance graph to represent the 'generation' relationship between activity p and entity d.

In addition to capturing usage and generation, additional details like configuration parameters and environment information (*e.g.,* CPU speed, memory capacity, network bandwidth) can be stored as attributes connected to the transformation. Inside each transformation, there can be number of intermediate tasks which may themselves generate intermediate data products. A MapReduce job for instance has multiple map and reduce tasks. Capturing provenance from such internal tasks at a high enough level to be useful helps in debugging and reproducing transformations.

When the output data from one analysis tool is used as the input to another, integration of provenance collected from both transformations can be guaranteed only by a consistent lake-unique persistent ID policy [6]. This may require a global policy enforced for all contributing organizations to a Data Lake. This unique ID notion could be based on file URLs and randomly generated data identifiers which are appended to data records when producing outputs so that the following transformations can use the same identifiers. It could also be achieved using globally persistent IDs such as the Handle system or DOIs. As a simple example, consider Figure 2a. The data product d_1 is subject to transformation T_1 and generates d_2 and d_3 as results. T_2 uses d_3 together with a new data product d_4 and generates d_5, d_6 and d_7. Finally T_3 uses d_6 and d_7 and generates d_8 as the final output. When all three transformations T_1, T_2 and T_3 have sent provenance events, a complete provenance graph is created in the central provenance collection system. Figure 2b shows the high level data lineage graph which represents the data flow starting from d_8.

D. Reference Architecture

The reference architecture, shown in Figure 3, uses a central provenance collection subsystem. Provenance events captured from components in the Data Lake are streamed into the provenance subsystem where they are processed, stored and analysed. The Provenance Stream Processing and Storage component at the heart of this architecture accepts the stream of provenance notifications (Ingest API) and supports queries (Query API). A live stream processing subsystem supports live queries while storage subsystem persists provenance for long term usage. When long running experiments in the Data Lake produce large volumes of provenance data, stream processing techniques become extremely useful as storing full provenance is not feasible. The Messaging System guarantees reliable provenance event delivery into the central provenance processing layer. Usage subsystem shows how provenance collected around the Data Lake can be used for different purposes. Both live and post-execution queries over collected provenance with Monitoring and Visualization helps in scenarios like the two use cases that we discussed above. There are other advantages as well such as Debugging and Reproducing experiments in the Data Lake.

In order to capture information about the origins of data, provenance must be captured at the Ingest. Some data products may carry their previous provenance information which should be integrated as well. Researchers may export data products from the Data Lake in some situations. Such data products should be coupled with their provenance for better usage.

IV. Prototype Implementation

We set up a prototype Data Lake and implemented a use case on top of it to evaluate the feasibility of our reference architecture. We used our provenance collection tools to capture, store, query and visualize provenance in our Data Lake. The reference architecture introduces both stored provenance processing and real time provenance processing for Data Lakes. In this prototype, we implement stored provenance

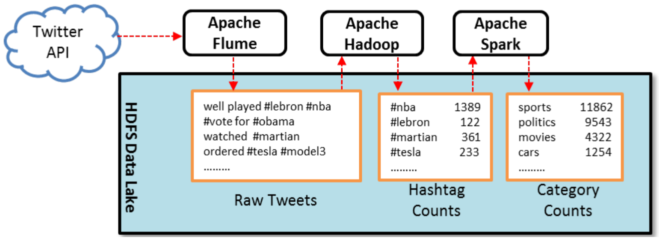


Fig. 4. Data Lake use case

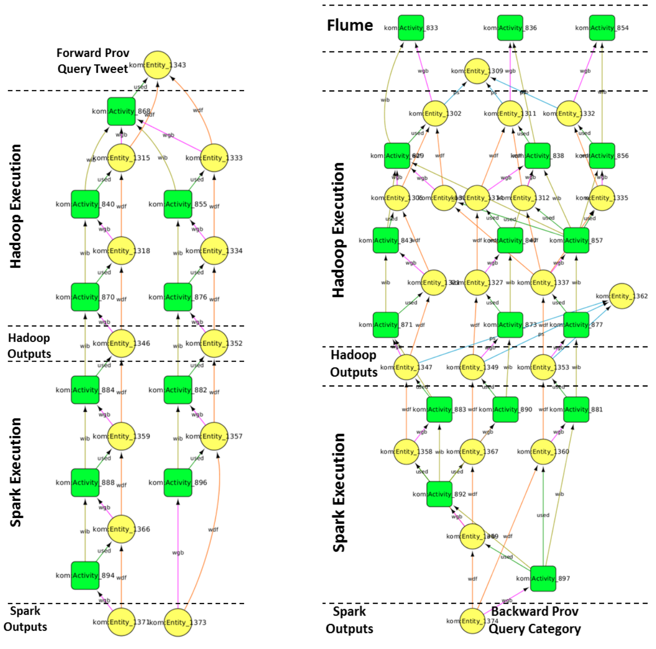
processing; real time provenance processing is future work. The central provenance subsystem uses our Komadu [15] provenance collection framework.

A. Komadu

Komadu is a W3C PROV based provenance collection framework which accepts provenance from distributed components through RabbitMQ[4] messaging and web services channels. It does not depend on any global knowledge about the system in a distributed setting. This makes it a good match for a Data Lake environment where different systems are used to perform different data transformations. Komadu API can be used to capture provenance events from individual components of the Data Lake. Each ingest operation adds a new relationship (`R`) between two nodes (a node can be an activity(`A`), entity(`E`) or agent(`G`)) of the provenance graph being generated. For example, when an activity `A` generates an entity `E`, the `addActivityEntityRelationship(A, E, R)` operation can be used to add a `wasGeneratedBy` relationship between `A` and `E`. Using the query operations, full provenance graphs including all connected edges can be generated for Entities, Activities and Agents by passing the relevant identifier. Backward only and forward only provenance graphs can be generated for Entities. In addition to that, Komadu API consists of operations to access the attributes of all types of nodes. Komadu Cytoscape[5] plugin can be used to visualize and navigate through provenance graphs.

B. Data Lake Use Case

The Data Lake prototype was implemented using an HDFS cluster and the transformations were performed using Hadoop and Spark. Analysing data from social media to identify trends is commonly seen in Data Lakes. As shown in Figure 4, we have implemented a chain of transformations based on Twitter data to first count hash tags and then to get aggregated counts based on categories. Apache Flume[6] was used to collect Twitter data and store in HDFS through the Twitter public streaming API. For each tweet, Flume captures the Twitter handle of the author, time, language and the full message and writes a record into an HDFS file. After collecting Twitter data over a period of five days, a Hadoop job was used to count hash tags in the full Twitter dataset. A new HDFS file with hash tag counts is generated as the result of the first Hadoop job which is used by a separate Spark job to get aggregated counts according to categories (sports, movies, politics etc). We just used a fixed set of categories for this prototype implementation to make it simple. In real Data Lakes, these transformations can be performed by different scientists at different times. They may use frameworks based on their preference and expertise. That is why we used two different frameworks for the transformations in our prototype to show how provenance can be integrated across systems.



(a) Forward Provenance

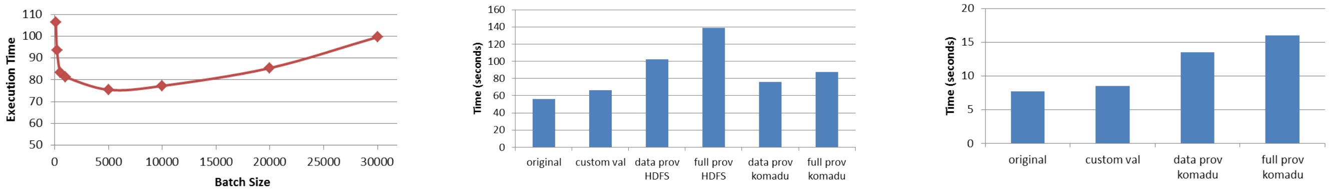
(b) Backward Provenance

Fig. 5. Forward and Backward Provenance

Komadu and its tool kit was used to build the provenance subsystem (shown in Figure 3) in our prototype. Komadu supports RabbitMQ messaging system and includes tools to fetch provenance notifications from RabbitMQ queues. A RabbitMQ instance was deployed in front of our Komadu instance so that all provenance notifications generated by Flume, Hadoop and Spark goes through a message queue in RabbitMQ. Ingested provenance events are asynchronously processed by Komadu and stored in relational tables. Stored provenance remains as a collection of edges until a graph generation request comes in. This delayed graph generation leads to efficient provenance ingest with minimum back pressure. This helps in a Data Lake environment where high volumes of provenance are generated.

To assign consistent identifiers for data items in our Data Lake, we followed the practice of appending identifies to data records when output data is written to the Data Lake. Subsequent transformation uses the same identifiers for provenance collection. Provenance events were captured in our prototype by instrumenting the application code that we implemented for each transformation. Tweet capturing code in Flume was instrumented to capture provenance at the data ingest into the Data Lake. *Map* and *Reduce* functions in the Hadoop job and *MapToPair* and *ReduceByKey* functions in the Spark job were instrumented to capture provenance from transformations. We

[4] http://www.rabbitmq.com/
[5] http://www.cytoscape.org/
[6] https://flume.apache.org/



(a) Execution time with varying batch size

(b) Hadoop Execution times for different scenarios

(c) Spark Execution times for different scenarios

Fig. 6. Experiments based on prototype Data Lake

implemented a client library with a simple API (like Log4J API for logging) which can be used to easily instrument Java applications for provenance capture. It minimizes the provenance capturing overhead by using a dedicated thread pool to asynchronously send provenance events into the provenance subsystem. In addition to that, the client library uses an event batching mechanism to minimize the network overhead by reducing the number of messages sent into the provenance subsystem over the network.

C. Provenance Queries and Visualization

After executing the provenance enabled Hadoop and Spark jobs on collected Twitter data, Komadu query API was used to generate provenance graphs. Komadu generates PROV-XML provenance graphs and it comes with a Cytoscape plugin which can be used to visualize and explore them. Fine-grained provenance includes input and output datasets for each transformation, intermediate function executions and all intermediate data products generated during the execution. Provenance from Flume, Hadoop and Spark have been integrated together through the usage of unique data identifiers.

Figure 5 shows forward and backward provenance graphs generated for a very small subset of tweets. Forward provenance is useful to derive details about the usages of a particular data item. Figure 5a shows a forward provenance graph for a single tweet. It shows the hash tags generated by that particular tweet in Hadoop outputs and the categories to which those hash tags contributed in Spark outputs. A backward provenance graph starting from a category under Spark outputs is shown in Figure 5b. This graph can be used to find all tweets which contributed for that category. For example, if a scientist wanted to get an age distribution of the authors who tweeted about sports, it can be done by finding the set of Twitter handles of the authors through backward provenance.

D. Performance Evaluation

To build our prototype, we used five small VM instances with 2 CPU cores of 2.5GHz speed, 4 GB of RAM and 50 GB local storage on each instance. Four instances were used for the HDFS cluster including one master node and three slave nodes. Total of 3.23 GB Twitter data was collected over a period of five days by running Flume on the master node. Hadoop and Spark clusters were set up on top of our four node HDFS cluster. One separate instance was allocated to set up the provenance subsystem using RabbitMQ and Komadu tools.

In order to minimize the provenance capture overhead, we used a dedicated thread pool and a provenance event batching mechanism in our client library. When the batch size is set to a relatively large number (>500), execution time becomes almost independent of the thread pool size as the number of messages sent through the network reduces. Therefore, we set the client thread pool size to 5 in each of our experiments. Figure 6a shows how the provenance enabled Hadoop execution time for a particular job varies when the batch size is increased from 100 to 30000 (provenance events). As per this result, we set the batch size to 5000 in each of our experiments. We used JSON format to encode provenance events and the average event size is around 120 bytes. The average size of a batched message sent over the network is around 600 KB (5000 x 120 bytes).

Figure 6b shows the execution times of the Hadoop job for different scenarios. Column 'original' represents the Hadoop execution time without capturing any provenance. In order to relate `Map` and `Reduce` provenance, we had to use a customized value field (in key-value pair) which contains data identifiers like in Ramp [10]. As shown by 'custom val' column in the chart, usage of customized value introduces an overhead of 19.28% and that is included in all other cases. Execution overhead depends on the granularity of provenance as well. Columns 'data prov komadu' and 'full prov komadu' shows the execution times of Hadoop when our technique is used to capture provenance. Data provenance (data relationships only) case adds a 36.47% overhead while full (data and process relationships) provenance case adds a 56.93% overhead. Table I shows a breakdown of provenance sizes generated for each case in Hadoop for the input size of 3.23 GB. Size of provenance doubles for full provenance case compared to data provenance and that leads to greater capturing overheads. As it is a common practice [10] to write provenance into HDFS in Hadoop jobs, we modified the same Hadoop job to store provenance events in HDFS as well and compared the overhead with our method. As shown by 'data prov HDFS' and 'full prov HDFS' columns in Figure 6b, that adds larger overheads compared to our techniques. Better performance have been achieved by modifying or extending Hadoop [11]. But our techniques operate completely on application level without modifying existing frameworks.

Figure 6c shows the execution times for the Spark job for different scenarios. Like in Hadoop, we used a customized output value to include data identifiers in Spark as well. That adds an overhead of 7.5% compared to original execution

TABLE I
SIZE (IN GB) OF PROVENANCE GENERATED BY HADOOP

	Map	Combine	Reduce	Total
Data Provenance	3.232	1.281	0.529	5.042
Full Provenance	9.733	1.824	0.813	12.37

time as shown by 'custom val' column. Data provenance and full provenance cases using Komadu add overheads of 76.1% and 108.35% respectively. Overhead percentages added by provenance capture in Spark is larger compared to Hadoop as Spark works faster than Hadoop and our techniques introduce same level of overhead in both cases.

V. RELATED WORK

Apache Falcon[7] manages the data lifecycle in Hadoop Big Data stack. Falcon supports creating lineage enabled data processing pipelines by connecting Hadoop based processing systems. Apache Nifi[8] is another data flow tool which captures lineage while moving data among systems. Neither tool captures detailed provenance within transformation steps (like in Figure 5). Few recent studies target provenance in individual Big Data processing frameworks like Hadoop and Spark. Wang J. et al. [9], [16] present a way of capturing provenance in MapReduce workflows by integrating Hadoop into Kepler. Ramp [10] and HadoopProv [11] are two attempts to capture provenance by extending Hadoop. Provenance in Apache Spark [12] and provenance in streaming data [17] have also been studied. Capturing provenance in traditional scientific workflows [18], [19] is another area which has been studied in depth. None of these studies focus on integrating provenance from different frameworks in a shared environment. While any of these Big Data processing frameworks can be connected to a Data Lake, as we argued above, a Data Lake can not depend on such framework specific provenance collection mechanisms due to provenance integration challenges. Therefore, provenance stitching [13] techniques are hard to apply. Wang, J. et al. [20] identify the challenges in Big Data provenance which are mostly applicable in a Data Lake environment. Distributed Big Data provenance integration has been identified as a challenge in their work where we present a solution in the context of a Data Lake.

VI. CONCLUSION AND FUTURE WORK

The reference architecture for integrated provenance demonstrates early value of data provenance as a lightweight approach to traceability. Future work addresses the viability of the approach in obtaining necessary information without excessive instrumentation or manual intervention. Scalability of the technique is to be further assessed within a real Data Lake environment. Persistent ID solutions have tradeoffs; the suitability of one over another in the Data Lake setting is an open question. We have implemented only the stored provenance processing techniques in the presented prototype. But our reference architecture highlights the power of real-time or live provenance processing in Data Lakes which is also left as a future work.

[7] http://falcon.apache.org/
[8] http://nifi.apache.org/

ACKNOWLEDGMENT

This work is funded in part by a grant from the National Science Foundation, ACI-0940824.

REFERENCES

[1] B. Stein and A. Morrison. The enterprise data lake: Better integration and deeper analytics. [Online]. Available: https://www.pwc.com/us/en/technology-forecast/2014/cloud-computing/assets/pdf/pwc-technology-forecast-data-lakes.pdf

[2] I. Terrizzano, P. M. Schwarz, M. Roth, and J. E. Colino, "Data wrangling: The challenging journey from the wild to the lake." in *CIDR*, 2015.

[3] M. Chessell, F. Scheepers, N. Nguyen, R. van Kessel, and R. van der Starre. (2014) Governing and managing big data for analytics and decision makers. [Online]. Available: http://www.redbooks.ibm.com/redpapers/pdfs/redp5120.pdf

[4] How to design a successful data lake. [Online]. Available: http://knowledgent.com/whitepaper/design-successful-data-lake/

[5] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. V. den Bussche, "The open provenance model core specification (v1.1)," *Future Gener. Comput. Syst.*, vol. 27, no. 6, pp. 743–756, Jun. 2011.

[6] L. Moreau and P. Groth, "Provenance: an introduction to prov," *Synthesis Lectures on the Semantic Web: Theory and Technology*, vol. 3, no. 4, pp. 1–129, 2013.

[7] I. Suriarachchi and B. Plale, "Provenance as essential infrastructure for data lakes," in *IPAW*. Springer, 2016.

[8] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: an extensible system for design and execution of scientific workflows," in *SSDBM*, June 2004, pp. 423–424.

[9] J. Wang, D. Crawl, and I. Altintas, "Kepler+hadoop: A general architecture facilitating data-intensive applications in scientific workflow systems," in *WORKS*. ACM, 2009, pp. 12:1–12:8.

[10] H. Park, R. Ikeda, and J. Widom, "Ramp: A system for capturing and tracing provenance in mapreduce workflows," in *VLDB*. Stanford InfoLab, August 2011.

[11] S. Akoush, R. Sohan, and A. Hopper, "Hadoopprov: Towards provenance as a first class citizen in mapreduce," in *TaPP*. USENIX Association, 2013, pp. 11:1–11:4.

[12] M. Interlandi, K. Shah, S. D. Tetali, M. Gulzar, S. Yoo, M. Kim, T. D. Millstein, and T. Condie, "Titian: Data provenance support in spark," *PVLDB*, vol. 9, no. 3, pp. 216–227, 2015.

[13] P. Missier, B. Ludascher, S. Bowers, S. Dey, A. Sarkar, B. Shrestha, I. Altintas, M. Anand, and C. Goble, "Linking multiple workflow provenance traces for interoperable collaborative science," in *WORKS*, Nov 2010, pp. 1–8.

[14] W. Oliveira, D. de Oliveira, and V. Braganholo, "Experiencing prov-wf for provenance interoperability in swfmss," in *IPAW*. Springer, 2014, pp. 294–296.

[15] I. Suriarachchi, Q. Zhou, and B. Plale, "Komadu: A capture and visualization system for scientific data provenance," *Journal of Open Research Software*, vol. 3, no. 1, 2015.

[16] D. Crawl, J. Wang, and I. Altintas, "Provenance for mapreduce-based data-intensive workflows," in *WORKS*. ACM, 2011, pp. 21–30.

[17] W. Sansrimahachai, L. Moreau, and M. Weal, "An on-the-fly provenance tracking mechanism for stream processing systems," in *ICIS*, June 2013, pp. 475–481.

[18] I. Altintas, O. Barney, and E. Jaeger-Frank, "Provenance collection support in the kepler scientific workflow system," in *Provenance and annotation of data*. Springer, 2006, pp. 118–132.

[19] Y. Simmhan, B. Plale, and D. Gannon, "A framework for collecting provenance in data-centric scientific workflows," in *ICWS*, Sept 2006, pp. 427–436.

[20] J. Wang, D. Crawl, S. Purawat, M. Nguyen, and I. Altintas, "Big data provenance: Challenges, state of the art and opportunities," in *Big Data*, Oct 2015, pp. 2509–2516.