SEeSAW - Similarity Exploiting Storage for Accelerating Analytics Workflows

Kalapriya Kannan, Suparna Bhattacharya, Kumar Raj, Muthukumar Murugan, Doug Voigt
{*kalapriya.kannan, suparna.bhattacharya, kumar.raj, muthukumar.murugan, doug.voigt*}*@hpe.com*
Hewlett Packard Enterprise

Abstract

The key to successful deployment of big data solutions lies in the timely distillation of meaningful information. This is made difficult by the mismatch between volume and velocity of data at scale and challenges posed by disparate speeds of IO, CPU, memory and communication links of data storage and processing systems. Instead of viewing storage as a bottleneck in this pipeline, we believe that storage systems are best positioned to discover and exploit intrinsic data properties to enhance information density of stored data. This has the potential to reduce the amount of new information that needs to be processed by an analytics workflow. Towards exploring this possibility, we propose ***SEeSAW***, a **S**imilarity **E**xploiting **S**torage for **A**ccelerating Analytics **W**orkflows that makes similarity a fundamental storage primitive. We show that ***SEeSAW*** transparently eliminates the need for applications to process uninformative data, thereby incurring substantially lower costs on IO, memory, computation and communication while speeding up (about 97% as observed in our experiment) the rate at which actionable outcomes can be derived by analyzing data. By increasing capacity of analytics workloads to absorb more data within the same resource envelope, **SEeSAW** can open up rich opportunities to reap greater benefits from machine and human generated data accumulated from various sources.

1 Introduction

Ease of access to numerous sources of data creates an abundance of rich historical context, generating opportunities to benefit from timely insights derived by analytics on that data. Our ability to reap those benefits remains limited only by our capacity to absorb that information. Actual performance on the ground has remained far from expectations as scales grow, largely attributed to the disparate speeds of the subsystems: the IO, CPU, memory, storage and communication links. In particular, with speeds far lower than other subsystems, storage is considered the long pole for analytics workloads. This creates a dilemma of not being able to utilize all the data context even when it is easily available.

What if we turn storage into an advantage for big data processing by exploiting its proximity to data and leverage useful data properties? Similarity is one such property underpinning the most prevalent principles utilized at different layers of data processing systems [2]. Making similarity recognition a fundamental primitive in storage enables a new class of storage systems which we call Similarity Exploiting Storage (SEeSAW) with several advantages to list a few: (a) workflows on similar data sets are accelerated (b) common computations on similar data sets by different analytical applications can be observed and used to prevent generation of nearly redundant results (c) new storage optimization opportunities can be evolved by assigning lower value to datasets that are similar or that result in similar analysis outcomes (d) data movement to perform similarity detection at higher layers is reduced and (e) analytical applications and frameworks can operate without modification.

We derive motivation from our own observations of a forecasting engine called Cloud Optimizer [1] that predicts resource demands. We find that 98% of the times the datasets collected from several thousands of VM machines every 3 days shows little or no variation in trends. Yet, the data is processed repeatedly by the same analytics application, each execution lasting for long durations (roughly 20 hours/per execution). A similarity detection algorithm will significantly reduce resource consumption by bypassing needless computation. We observe that checking for partially identical data (such as sliding window hashes in MapReuse [9]) would miss opportunities for detecting semantically or statistically similar patterns. For example, a time series might consist of a trend component, a cyclical component [5] and a remainder. The trends or cycles could remain the same even when individual data samples exhibit significant variation. In this paper, we show that for certain categories of analytics applications, such as forecasting and prediction (e.g. weather prediction, resource utilization forecasting, website load forecasting and predictive log analytics) ***SEeSAW*** can yield huge gains. For example, we observe up to a 97% speed-up using similarity detection at storage to accelerate the Cloud Optimizer workflow without making any changes to the application.

2 Details of SEeSAW System

2.1 Overview

Analytical workflows comprise a sequence of steps that perform certain actions on ingested data (referred to as the Extract Transform Load stage) and a model learning or scoring stage. At each stage there is an output (insight) (referred to as Transformation Output (TO)) that is generated. **SEeSAW** transparently intercepts the workflow to allow comparison of output that is generated in any of these stages (TO) with the data generated by an earlier execution at the same stage.[1] Presence of a similar TO at a stage indicates that the execution of all following stages could be redundant as the Analytics Output (*AO*) is likely to be nearly the same as the earlier execution. In such cases, if **SEeSAW** detects a previous TO that is similar, it can directly utilize results from earlier executions. All the stages following the stage at which similarity was detected can be bypassed.

Similarity measurements utilize distance metrics between data sets and differ depending on the type of the data. For example, on images Locality Sensitive Hashing (LSH) can be used and on text data, Jaccard or Cosine similarity metrics can be used. With **SEeSAW** TOs are compared rather than the raw inputs. This gives **SEeSAW** the advantage of operating with a smaller set of standard data similarity algorithms. Distilled intermediate results typically belong to a smaller subset of the variety of data formats. Let us consider an example of two workflows–one that ingests text documents and another that takes images. Similarity analysis on text would require key word matching, and similarity analysis on images would require Locality Sensitive Hashing (LSH). However, intermediate steps in the workflow can convert textual input to numerics such as topic distribution and segmented image data into statistics of key visual features. Similarity algorithms on numerics (eg., correlation coefficient) will suffice in both cases. Further, as we move to later stages in the analytics execution pipeline, the TO is likely to contain features that are more relevant for the analytics model(e.g. sentiment polarity, discrete Fourier transform components). Hence a distance metric at these stages is more likely to reflect semantic similarity compared to the raw data.

[1]The technique can be generalized for stages having multiple TO and input.

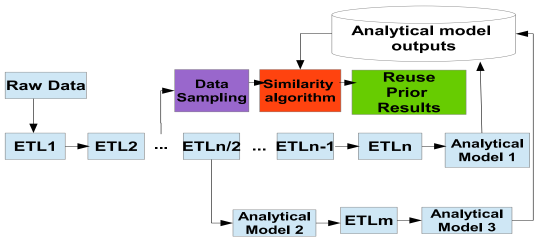


Figure 1: Transformation and Similarity detection.

Figure 1 shows the different stages of an Analytics workflow and the functional role of *SEesAW*. A similarity algorithm could be plugged into **SEeSAW** and invoked at specific stages to perform similarity analysis. Once similarity is established **SEeSAW** can transparently bypass the remaining stages while making the results of the previous execution available for the newly ingested data.

2.2 Goals

We enumerate the goals of **SEeSAW**:

Lightweight Mechanism for Data Similarity Utilization: Exploit similarity in data with minimal overhead for similarity detection. Identify the most suitable stages for performing similarity tests by taking into account the impact of TO on the analytics model.

Reduce Time to get analytics results: Accelerate the time taken to derive the *AO* from ingested data by exploiting the presence of similarity.

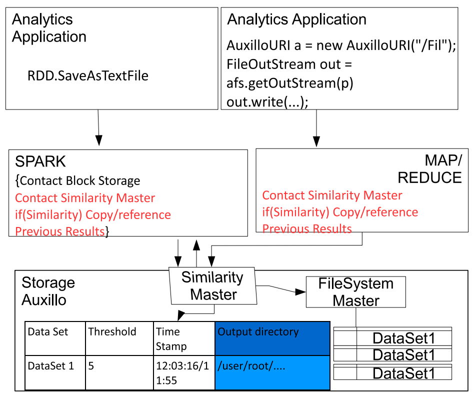
Save IO, CPU, Communication and Storage Space: Consumption of CPU, network and storage resources is reduced to enable more data processing. With information about similarity *SEeSAW*, enables storage to evolve better data tiering and space optimization policies.

Seamless infusion of similarity algorithm and outcomes: Transparent detection of similarity and reuse existing *AO* seamlessly. Analytical applications and frameworks should be agnostic of the underlying similarity exploiting system and hence need not be modified in order to work with *SEeSAW*.

2.3 SEeSAW Architecture

Storage exposes a Similarity Primitive (SP) that can operate on data to execute a similarity algorithm. The SP considers other datasets from previous executions of the same workflow. Figure 2 shows our prototype similarity service that is currently implemented as a service in Alluxio [7].[2] Data is written to Alluxio through specific

[2]It should be noted that similarity checking can be performed completely at the storage. The storage can exploit similarity results as part



```
RDD.SaveAsTextFile
```

```
AuxilloURI a = new AuxilloURI("/Fil");
FileOutStream out =
afs.getOutStream(p)
out.write(...);
```

```
{Contact Block Storage
Contact Similarity Master
if(Similarity) Copy/reference
Previous Results}
```

```
Contact Similarity Master
if(Similarity) Copy/reference
Previous Results
```

Data Set	Threshold	Time Stamp	Output directory
DataSet 1	5	12:03:16/1 1:55	/user/root/....

Figure 2: Similarity in Spark Alluxio framework

write calls.

Central to *SEeSAW* is the meta-data that manages the information about the existing datasets in the Alluxio File System. The metadata includes an identity for each dataset, similarity threshold metrics (the bounds within which two pieces of data can be considered to be similar), the time-stamp for versioning and the location of the *AO* from the earlier execution. This meta-data is utilized by the similarity service to test for similarity when new data is ingested.

The Similarity Service performs two different functions: (a) Optimal Stage Analysis (OSA) and (b) Similarity Computation (SC). In the OSA stage, we compute the optimal point in the workflow to run the similarity computation. This requires information about the different stages of the analytical functions. We use lineage tracking to provide this information to the *SP*. For a new dataset and a new analytics program, all write calls (i.e., TO) are passed to the file system. This along with the lineage graph is used to compute the stage at which similarity detection can be performed. In SC similarity detection is performed on the identified stage. If there exists an earlier TO that closely matches data from the current execution, we terminate the current workflow and present the corresponding *AO* from the previous run.

In using **SEeSAW**, there is a storage space requirement for retaining TOs from the previous execution. This is necessary only for the optimal stage where similarity check is performed. Policies such as LRU may be used to manage the TOs saved so that *SEeSAW* maintain and index the most relevant non-similar representatives from all previous executions.

of any call that requests data. As a radical approach we are exploring the feasibility of building similarity primitive as part of logical read and write calls to the storage from an analytics framework, largely eliminating changes to the Spark framework.

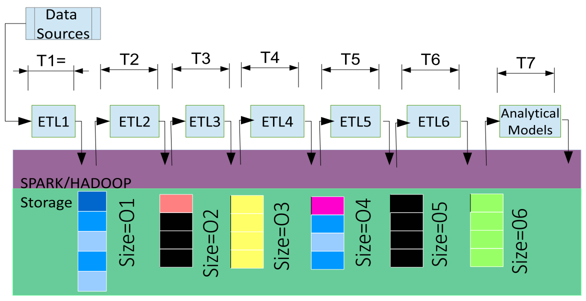


Figure 3: Different stages in analytics with time

2.4 Optimal Stage Analysis

We consider typical analytical workflows which consist of many stages, each ingesting a dataset and producing a TO. Figure 3 shows the different stages and the sizes of data output at each stage.

The criteria used to determine the optimal stage for similarity detection in the workflow execution pipeline is dictated by two parameters: (1) the remaining time to the final output (AO) i.e., the potential saving our method can provide by bypassing the remaining stages and (2) the time to compute similarity which in turn is determined by the volume of data output at each stage. The earlier we detect similarity in the pipeline, the bigger the benefit achieved by bypassing the steps to obtain the end result. However, at the later stages in the execution pipeline, the size of the transformed data to test for similarity is usually a lot smaller (e.g. a distilled feature set). The remaining time to final output as well as the similarity detection cost is estimated during the OSA phase based on previous runs of the same application. For OSA, we need to record the time for all stages of the analytics workflow along with the similarity computation. This is performed only for the first few executions for a given analytics workflow. Optimally trading off the time saving against the overhead of similarity detection determines the stage to use for similarity detection in subsequent executions. The Similarity Threshold (ST) is then adjusted to bound the acceptable distance between *AO*'s across executions where data sets exhibit similarity at the optimal stage.

3 Evaluation and Assessment

For the purpose of experimentation, we implemented the **SEeSAW** similarity primitive as part of the Alluxio file system services and modified Spark to use SP to bypass workflows if similarity is detected at the optimal stage, as shown earlier in Figure 2.

Dimensionality-reduction techniques[4] [11] can be incorporated into the **SEeSAW** data sampling phase.

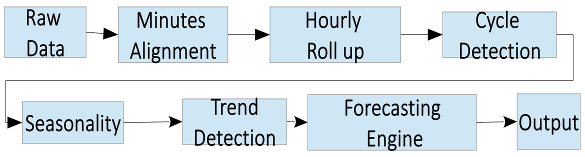


Figure 4: Analytics workflow for the VM utilization forecasting engine.

	$(ST) \leq 5$	$(ST) \leq 8$	$(ST) \leq 10$
Percentage dataset	96.5%	98%	98.9%

Table 1: % similarity found in the specified threshold.

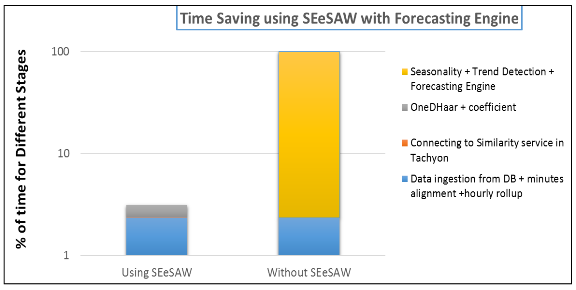


Figure 5: Time taken by analytics workflow with and without **SEeSAW**.

This phase transforms TO data into a compact summarized representation so that it can be used to efficiently perform similarity comparisons. For this paper, we implemented a specific type of discrete wavelet transform known as Haar Wavelet Transform for dimensionality reduction of time series data. Details of the Haar Wavelet Transform can be obtained from [3].

Dataset: We consider a real world analytical workflow that utilizes time-series data to evaluate the benefits of **SEeSAW**. We use a commercial forecasting engine for cloud provider VMs[1]. Resource utilization of VM instances in a cloud environment is monitored and used for forecasting utilization for the next 90 days. The data collected every three days is used to detect pattern changes. If there is no change in the pattern, the forecast value is retained, otherwise the models are retrained for new forecasting. We show through experiments two important results, that (a) there is high similarity, with little or no variation between these datasets utilized every 3 days (b) Speedup by about 95% of the time to obtain the AO when we employ **SEeSAW**.

The forecasting engine collects dataset from about 1500 instances of VM and the parameters monitored are memory, storage and CPU utilization (roughly about 1.5 GB of data). The data is collected for every 5 mins (12 data points for each hour) for a period of 90 days. The forecasting engine predicts the utilization of the 1500 instances for the next 90-day. If there are outliers or utilizations predicted beyond or nearing the available capacity then appropriate action is suggested to the administrators and VM migrations are initiated.

Figure 4 shows the different stages of the analytical programs. The data is directly read from a data store (an HPE Vertica database) into a Spark RDD and various transformations are applied. For OSA, we perform at least one complete execution of the forecasting engine. The critical path consisting of the path that takes the longest time is considered. For sake of brevity, we only present the observations i.e., the algorithm discovers that the hourly roll up stage has the highest potential savings from similarity exploitation. Therefore, any subsequent execution of the analytics workflow can just perform similarity comparisons only for the output of this stage.

Results: We first present the results for the observed similarity in the dataset. We take the baseline strategy used by the forecasting engine once every 3 days to check for pattern change. For this experiment, we consider data monitored every 3 days (1-3 days, 1-6, 1-9 and so on) in the 90 days observation period. Distance measurement such as Pearsons or Corelation coefficient can be used. In our work, we compute the difference between the data points. If the difference is less than a threshold then we say that the data is similar. Table 1 shows the number of instances of similarity detected on the data and for different Similarity Threshold (ST).

Figure 5 shows the time that is spent on performing different operations for the same dataset. We partition the program execution in 3 parts: Part (a) the combined time of all stages before the “Hourly roll up”, and Part (b) time taken by the similarity computation (through the “save as” functionality), and Part (c) for all the steps after “hourly roll up”. Figure 5 shows the time taken for various stages. Speed-up obtained is about 97% compared to the time when the forecasting engine is executed without **SEeSAW**.

We also performed experiments to observe the overhead (in terms of time taken) to compute similarity when the number of datasets for similarity computation available are high. Figure 6 presents the estimates of time spent for computing similarity with different indexing schemes. All time is normalized with respect to the time observed for 100 instances using a brute force approach. It can be seen that even when we consider the time taken by the worst-case brute force approach, the benefits obtained (savings in the time taken for forecasting) are significant (where the new data is compared with all instances of the exiting datasets). We plan to explore methods for faster indexing as an area of future work.

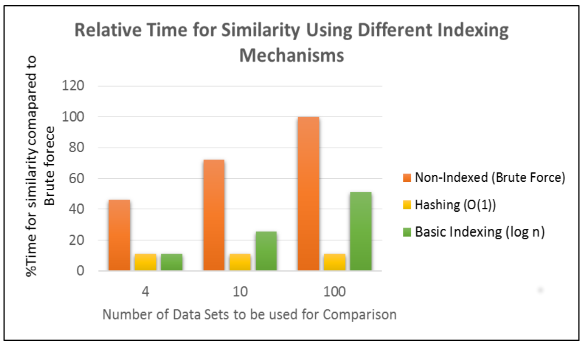


Figure 6: Graph showing the time taken for performing similarity operations.

4 Related Work

There is a significant body of work in the area of detecting identical data sets to reduce redundant computation[6][10][9]. Nectar [6] incorporates unifies data management with computation and reuses old results for portions of input data found identical to the previous execution. A transparent way of capturing both process and file lineage for program executions is proposed in [10]. Results are cached via the lineage mechanism and further queried to provide the cached results to repeated application queries. MapReuse [9] computes hashes of each block, and detects incremental changes by moving a sliding window byte by byte and computing the hash of each window This body of work uses the notion of identical or partially identical data sets, unlike SEeSAW where we utilize statistical or semantic similarity of the intermediate outputs.

To save space, ImpressionStore [12] does not store all data accurately but only what it is needed to ensure accuracy of results for specific class of applications which retrieve only the top-K components. In contrast, we propose similarity measures to seamlessly save computation on similar data and resulting outputs for any analytics workflow. Analyzethis [8] pushes an analytics workflow to lower layers of a storage system, making the system analytics-aware. Instead SEeSAW aims to operate transparently by exploiting data properties such as similarity in the form of storage level primitives.

5 Challenges and Open Questions

We have shown how a storage system can exploit similarity to offer promising benefits for analytics workloads. This is backed by our preliminary experiment on transparent acceleration of a real world forecasting workflow, besides the scope for sophisticated storage optimizations such as smart tiering and data reduction. This leads us to the broader question about other data properties storage systems could exploit.

Meanwhile, before storage systems can embrace similarity detection as a full-fledged reliable service, several speculations and open challenges need to be debated and addressed. To enumerate a few, (a) While content management systems have used similarity to cluster data, why haven't mainstream storage systems taken advantage of it? (b) Are there other use cases where similarity is relevant? (c) What is the appropriate choice of features and algorithms for measuring data similarity? How can it be generalized? (d) How much do similarity-based decisions affect accuracy of the analytics applications? (e) How many prior executions are required to be stored for comparison? Our early implementation of **SEeSAW** currently sidesteps many of these issues by restricting itself to certain specific data types/formats. In the long run, we envision **SEeSAW** will open doors to a promising world of analytics-friendly storage.

References

[1] Cloud optimizer hp cloud. `https://marketplace.hpcloud.com/cloudoptimizer`.

[2] AGARWAL, R., KHANDELWAL, A., AND STOICA, I. Succinct: Enabling queries on compressed data. In *12th Usenix NSDI* (2015), NSDI'15, USENIX Association.

[3] CUI, H., KEETON, K., ROY, I., VISWANATHAN, K., AND GANGER, G. R. Using data transformations for low-latency time series analysis. In *6th* (2015), SoCC '15, ACM.

[4] GILBERT, A. C., KOTIDIS, Y., MUTHUKRISHNAN, S., AND STRAUSS, M. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *27th VLDB* (2001), VLDB '01.

[5] GMACH, D., ROLIA, J., CHERKASOVA, L., AND KEMPER, A. Workload analysis and demand prediction of enterprise data center applications. IISWC '07, IEEE Computer Society.

[6] GUNDA, P. K., RAVINDRANATH, L., THEKKATH, C. A., YU, Y., AND ZHUANG, L. Nectar: Automatic management of data and computation in datacenters. In *9th USENIX* (2010), OSDI'10, USENIX Association.

[7] LI, H., GHODSI, A., ZAHARIA, M., SHENKER, S., AND STOICA, I. Tachyon: Reliable, memory speed storage for cluster computing frameworks. SOCC '14, ACM.

[8] SIM, H., KIM, Y., VAZHKUDAI, S. S., TIWARI, D., ANWAR, A., BUTT, A. R., AND RAMAKRISHNAN, L. Analyzethis: An analysis workflow-aware storage system. SC '15, ACM.

[9] TIWARI, D., AND SOLIHIN, Y. Mapreuse: Reusing computation in an in-memory mapreduce system. In *2014, 28th* (2014), IPDPS '14, IEEE Computer Society.

[10] VAHDAT, A., AND ANDERSON, T. Transparent result caching. In *USENIX ATC* (1998), ATEC '98, USENIX Association.

[11] WU, Y.-L., AGRAWAL, D., AND EL ABBADI, A. A comparison of dft and dwt based similarity search in time-series databases. CIKM '00, ACM.

[12] ZHANG, J., YAN, Y., CHEN, L. J., WANG, M., MOSCIBRODA, T., AND ZHANG, Z. Impression store: Compressive sensing-based storage for big data analytics. HotCloud'14, USENIX Association.