BRAID
A Hybrid Processing Architecture for Big Data

Corinna Giebler[1], Christoph Stach[1], Holger Schwarz[1], and Bernhard Mitschang[1]
[1]*Institute for Parallel and Distributed Systems, University of Stuttgart,*
Universitätsstraße 38, D-70569 Stuttgart, Germany
{giebleca | stachch | hrschwar | mitsch}@ipvs.uni-stuttgart.de

Keywords: Big Data, IoT, Batch Processing, Stream Processing, Lambda Architecture, Kappa Architecture.

Abstract: The *Internet of Things* is applied in many domains and collects vast amounts of data. This data provides access to a lot of knowledge when analyzed comprehensively. However, advanced analysis techniques such as predictive or prescriptive analytics require access to both, history data, i. e., long-term persisted data, and real-time data as well as a joint view on both types of data. State-of-the-art hybrid processing architectures for big data—namely, the *Lambda* and the *Kappa Architecture*—support the processing of history data and real-time data. However, they lack of a tight coupling of the two processing modes. That is, the user has to do a lot of work manually in order to enable a comprehensive analysis of the data. For instance, the user has to combine the results of both processing modes or apply knowledge from one processing mode to the other. Therefore, we introduce a novel hy**b**rid p**r**ocessing **a**rchitecture for b**i**g **d**ata, called *BRAID*. BRAID intertwines the processing of history data and real-time data by adding communication channels between the batch engine and the stream engine. This enables to carry out comprehensive analyses automatically at a reasonable overhead.

1 INTRODUCTION

In the *Internet of Things* (*IoT*), everyday objects are interconnected. Various sensors built into these objects capture the current context. This knowledge can be shared with any other connected *Thing*, i. e., a device equipped with sensors and connectivity options. These Things are used in various domains to optimize processes. The examples presented in this work focus on the Industry 4.0 as this domain is one of the key profiteers from the IoT (Middleton et al., 2013). Yet, the conclusions and results can be applied to any other big data use case.

By collecting a great amount of data over time, precise analyses provide insights into production processes, e. g., about the efficiency of a machine at the shop floor. At the same time, real-time analyses on data streams can be used to trigger automatic reactions in the case of an emergency, e. g., when a machine is overheating. Thereby, the efficiency and quality of the production process can be enhanced (Geissbauer et al., 2014).

A combination of both kinds of analyses provides even more insights, e. g., history production data can be analyzed to find patterns indicating an ideal production setting where only little rejects are produced. By analyzing real-time machine data, these patterns can be used as decision models to predict the quality of the workpieces, e. g., to stop the production process at an early stage if too many rejects would be produced.

As a consequence, mining algorithms such as the *VFDT* algorithm (Domingos and Hulten, 2000) operate on both, history data as well as real-time data. Initially, such an algorithm learns patterns from existing production data via batch processing. Then, it applies and refines these patterns using real-time production data via stream processing. Therefore, an underlying processing architecture has to support both modes of data processing. Current architectures strictly separate these processing modes, whereby such comprehensive analytics are hard to be realized. Yet, these analytics hold inestimable business value (Columbus, 2016). Therefore, we introduce a hybrid processing architecture, in which the processing of history and real-time data is tightly intertwined.

Thus, we make the following five contributions: (**a**) We describe an Industry 4.0 application scenario and identify required processing steps for big data analytics. (**b**) We analyze existing data processing architectures—namely, architectures for batch processing, stream processing, as well as the *Lambda* and *Kappa Architecture*—and assess their suitability for

comprehensive analytic processes. (**c**) We introduce a novel hy**b**rid p**r**ocessing **a**rchitecture for bi**g** **d**ata, called *BRAID*. Whereas state-of-the-art architectures have strictly isolated processing branches for history and real-time data, BRAID provides additional communication channels via which intermediate results from batch processing can be used in stream processing and vice versa. This enables comprehensive analytics. (**d**) We evaluate our architecture and compare it with the Lambda and Kappa Architecture. (**e**) We discuss implementation alternatives for BRAID.

The structure of this paper is as follows: An Industry 4.0 application scenario is given in Section 2. Section 3 presents different data processing approaches and assesses their usability. We introduce a novel architecture called BRAID in Section 4. We assess the applicability of BRAID to the introduced big data use case in Section 5. We discuss implementation alternatives in Section 6 before Section 7 concludes the paper.

2 APPLICATION SCENARIO

Underutilization of data generated during process execution is one of the key problems in current manufacturing IT systems (Gröger et al., 2014). By using advanced data analytic techniques, new knowledge about manufacturing processes can be gained from this data. For instance, current and future processes can be optimized by modifying process control at runtime. In the following, we characterize how a therefore required comprehensive analysis can be realized (see Figure 1).

The *Master Dataset* contains all data collected during a production process (1). This includes, e. g., throughput times or temperature changes in the workpieces. These highly heterogeneous data can be stored and managed in a dedicated data system such as *CoreDB* (Beheshti et al., 2017).

The long-term process data persisted in the Master Dataset is processed by a batch processing engine (2). The processing logic is implemented and configured by the administrator of the system. *Descriptive analytics* can be applied to the persisted data in order to condense the huge amount of raw data. Most business intelligence solutions support this kind of analytics. For instance, it is possible to reconstruct which data records led to a successful production result and the data can be annotated accordingly. Subsequently, *predictive analytics* can be used to make predictions for future production processes. For example, classification algorithms can operate on the pre-processed data and generate a decision tree based on the annotations.

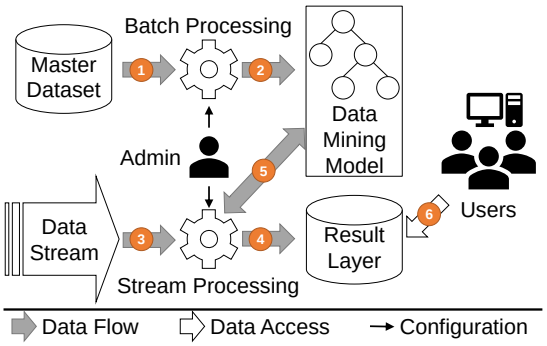

Figure 1: Comprehensive Analysis Steps.

The decision tree enables to predict which production times, temperatures, etc., lead to what production outcome. The decision tree is stored as a *Data Mining Model*.

However, to achieve a maximum business benefit from these models, they have to be applied to production data in real-time (Gröger et al., 2014). These data are emitted from the sensors installed in the production machines (3). They are processed by a stream processing engine (4).

In order to enable comprehensive analytics, decision trees generated on basis of the long-term process data have to be loaded into the stream processing engine as an initial configuration (5). Thereby, it is possible to detect production errors at an early stage and either initiate countermeasures or terminate subsequent production steps for an affected workpiece if a successful production result can no longer be achieved. The combined results of the comprehensive analytics process are stored in the *Result Layer* and made available for users or other systems (6).

However, as production conditions might alter over time, it is not appropriate to use a static decision tree model. Therefore, the results of the real-time analysis have to influence the model as well. Data mining algorithms such as VDFT (Domingos and Hulten, 2000) consider this. Here, the decision tree is adjusted to changing conditions. This means that not only a recommendation for action is calculated, but also the achieved outcome is evaluated. If some of the tree's parameters are no longer up to date, the stored model has to be changed (5). For instance, new raw materials might be less prone to temperature changes, whereby the temperature thresholds at which rejects are presumably produced have to be changed in the tree. It is evident that therefore a tight coupling between the two processing components is required. However, there is little software support for this kind of analytics as such methods have special requirements towards the underlying infrastructure. In the following, we look into

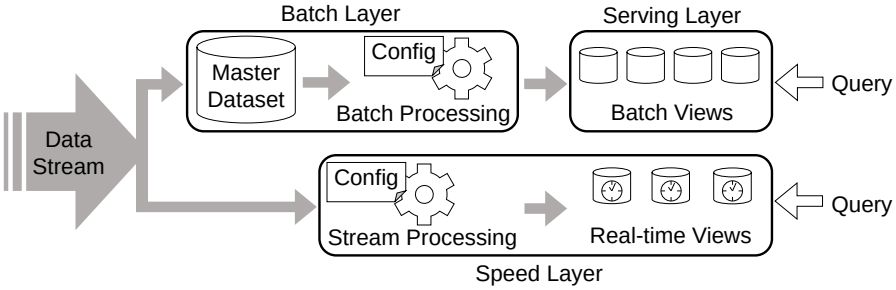

Figure 2: The Lambda Architecture (adapted from (Marz and Warren, 2015)).

current big data processing architectures and discuss their applicability.

3 RELATED WORK

As shown in the preceding section, two different types of data have to be considered in order to accomplish comprehensive analyses: On the one hand, persisted long-term process data are required to gain knowledge about past production processes in order to draw conclusions about future production processes. This kind of long-term data is labeled as *history data* in the following. As history data is gathered and stored over a long period of time, a processing system for this data has to be able to handle a large amount of data whereas the required processing time is of secondary importance. On the other hand, data which is sent directly from a data source (e. g., a sensor) to a processing sink without persisting it, has to be processed in order to analyze current production processes. This kind of data is labeled as *real-time data* in the following. As real-time data is only significant for a short period of time (e. g., a deviation from a reference value has to be detected as soon as it occurs to enable immediate countermeasures), a processing system for this data has to be able to handle it very fast. However, only a small amount of data has to be processed at each point in time.

This shows that different processing systems are required in order to process both, history data and real-time data. Literature differentiates between three alternative big data processing methods: batch processing (see Section 3.1), stream processing (see Section 3.2), and hybrid processing (see Section 3.3) (Casado and Younas, 2015). Hybrid approaches provide both, batch processing and stream processing. The Lambda Architecture (Marz and Warren, 2015) and the Kappa Architecture (Kreps, 2014) are state-of-the-art concerning hybrid big data processing. Section 3.4 discusses the applicability of the reviewed architectures for the realization of comprehensive analyses.

3.1 Batch Processing

Batch processing systems operate on large amounts of data which are stored persistently in a database or a file system. By processing this data, it is expected that data mining results will be most accurate due to the large data stock. A batch processing system splits the entire data volume into smaller subsets, the *batches*, and processes each on distributed computation nodes. Typically, processing takes long due to the large amount of data. In most cases, a result is only available after all batches have been processed. In addition, the processing logic cannot be adjusted after processing has started (Casado and Younas, 2015). For instance, *Apache Hive* can be used for batch processing (Barbierato et al., 2013).

Batch processing systems are mostly used to process history data. However, it is also possible to process real-time data via such a system. When incoming real-time data is buffered in a very small data store for a limited amount of time, this store can be used as input for the batch processing system. Thereby, all data can be processed as a single small batch and the results are available in near-real-time. Yet, this is not the intended purpose of a batch processing system.

3.2 Stream Processing

A stream processing system splits incoming data for processing as streams are infinite (Casado and Younas, 2015). The processing results are immediately available to the users. Thereby, the processing logic can be altered at any time and the changes are automatically applied to the subsequent data entries. *Aurora* is an example for a stream processing system (Abadi et al., 2003).

Stream processing systems are mostly used to process real-time data. Yet, any data source can be processed as a data stream. For this purpose, each stored data set is sent individually to a sink. In this way, history data can also be processed by a stream processing system (cf. the Kappa Architecture in Section 3.3.2).

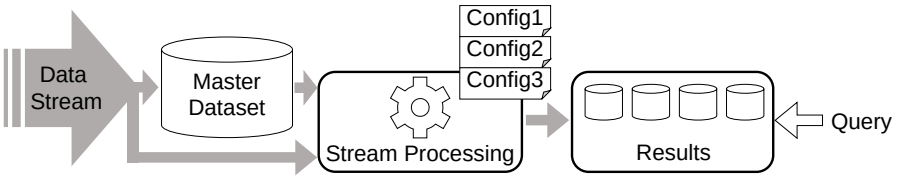

Figure 3: The Kappa Architecture (adapted from (Kreps, 2014)).

However, this kind of processing system is not meant to detect complex correlations within huge data stocks.

3.3 Hybrid Processing

Often both, high accuracy and low processing time are required. As neither batch nor stream processing can achieve this, hybrid processing combines both techniques. In the following, the Lambda and the Kappa Architecture are presented.

3.3.1 Lambda Architecture

The Lambda Architecture has separate branches for batch and stream processing (Marz, 2011). As shown in Figure 2, data is distributed to both branches. In the *Batch Layer*, data is added to an append-only data storage, the *Master Dataset*. Periodically, this data is processed using batch processing. Here, the raw data is prepared for the user. For instance, data mining is used to assess past production processes. The results are saved in so called *Batch Views*. Users can query these data via the *Serving Layer*.

However, batch processing operates only on data which is available in the Master Dataset when the processing is started. Since processing a large Master Dataset takes a lot of time, the batch views usually do not contain the most recent data. The *Speed Layer* addresses this issue. Here, data is processed in real-time by using stream processing. The results are stored in the *Real-time Views*.

Batch Views contain analytics results, which are based on a huge set of history data but without any information about the latest incoming data. Such information is provided as part of the Real-time Views, but only for a limited time frame. To obtain a complete view on the production process data, the user has to query both views and combine the results manually.

The two branches are strictly isolated from each other, i. e., each branch has its own processing logic, as indicated by the separate configuration files in Figure 2. The logic used in the Speed Layer can be adjusted quickly, while the Batch Layer is highly scalable. As a result, the schemata of the Batch Views and the Real-time Views can differ. Users have to consider this in their queries. Data access is therefore complex and time-consuming.

3.3.2 Kappa Architecture

The Kappa Architecture addresses this issue as it uses stream processing, only (Kreps, 2014). That is, incoming data is persisted in the Master Dataset and also processed immediately. The architecture is shown in Figure 3.

A key feature of the Kappa Architecture is the possibility to perform several separate stream processing jobs in parallel. This is indicated in Figure 3 by the several configuration files on the stream processing component. Due to this feature, the Kappa Architecture enables to process the entire Master Dataset, i. e., history data, in parallel to incoming real-time data. This way, history data can be processed again at any time with an alternative processing logic. This procedure can be used to perform a batch-like processing of history data via stream processing engines.

The advantage of the Kappa Architecture is its single data processing engine, which can be adjusted quickly. Yet, there is an increased demand concerning computational power and storage capacity when processing voluminous history data in parallel to real-time data streams.

3.4 Discussion

As shown in Section 2, there is a need for joint analytics of history and real-time data (for further use cases see (Gröger, 2018)). Yet, architectural support is required to enable all analytic steps. Three observations can be made in this respect based on the state of art: (**I**) Only hybrid architectures have the ability to process both, history and real-time data. (**II**) Only the Kappa Architecture has the ability to combine the results from batch and stream processing automatically. This makes data access considerably easier for users. (**III**) No current state-of-the-art approach enables to exchange intermediate results such as the data mining model in the given use case between the two processing engines. Yet, this is crucial to perform comprehensive analytics.

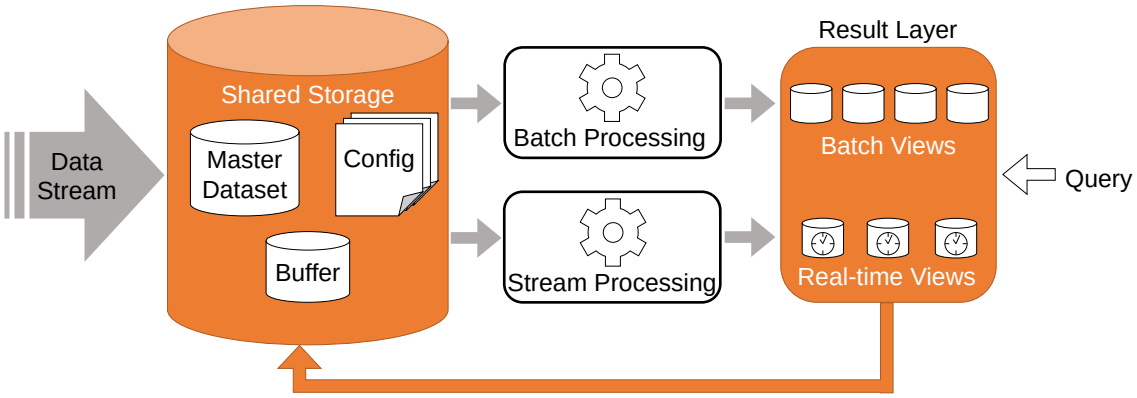

Figure 4: Architecture of BRAID (components that exceed state-of-the-art are depicted in orange).

4 THE BRAID APPROACH

Due to the aforementioned shortcomings of the state-of-the-art, a novel approach is required in order to enable comprehensive data analytics. Thus, we introduce BRAID (see Figure 4), which provides the required functionalities.

All incoming data is persisted in the Master Dataset. In BRAID, the Master Dataset is assigned to a *Shared Storage* available for both, batch and stream processing. Additionally, the latest data is also temporarily buffered for real-time processing. From there, the data is forwarded to one of the two processing branches (details on the two branches are given later). The Shared Storage is the key component towards comprehensive analytics as introduced in Section 2. For instance, intermediate results from batch processing (e. g., a data mining model) can be send back to the Shared Storage and used as input or configuration for the stream processing and vice versa.

After the processing, the results are stored in a shared *Result Layer*. Thereby, result sets can be stored in a joint data storage. Thus, the user can query the Result Layer directly, without considering which processing branch created the particular data. Different data schemata in the two result sets do not pose a problem for many storage systems such as *HDFS*[1]. The data can be annotated with structure metadata to facilitate data access (Quix et al., 2016).

The processing logic for both, batch and stream processing, is defined in configuration files which are stored in the Shared Storage. As both branches have read and write permissions for the Shared Storage, a common configuration file can be used to define the logic for both processing modes. In addition, results can be written back to the Shared Storage, e. g., to adjust the configuration at runtime. This enables flexible and application-oriented processing. The following paragraphs detail on the various aspects of the two processing branches.

The *Batch Processing Branch* is similar to the Batch and Serving Layer in the Lambda Architecture. Periodically, content of the Master Dataset is processed using batch processing, where configurations and results from the Shared Storage can be used. The results are stored in Batch Views in the Result Layer.

The *Stream Processing Branch* corresponds to the Speed Layer of the Lambda Architecture. Data is processed via stream processing. The use of the buffer is intended for real-time processing, as it ensures faster access times. By using data from the Master Dataset a Kappa-like behavior can be realized. In addition, access to the Shared Storage also enables to adjust the configuration files and write back results.

BRAID supports several processing modes. It can be used to emulate all architectures mentioned in Section 3. Additionally, the interconnection between the batch and the stream processing components due to the Shared Storage and the common Result Layer offers new processing possibilities, e. g., the realization of comprehensive analytics, as described in Section 2.

Batch and Stream Processing. Using the Master Dataset and BRAID's batch processing branch enables pure batch processing. Accordingly, pure stream processing can be realized by using the buffer and the stream processing branch.

Lambda Architecture. As BRAID is able to emulate both, batch and stream processing, also the Lambda Architecture can be emulated easily. To do so, real-time data is stored in the Master Dataset before being processed in the stream processing branch. The Master Dataset itself is processed in periodic intervals via batch processing. The Shared Storage is used to store the Master Dataset, and each of BRAID's

[1]The Hadoop Distributed File System (HDFS (Shvachko et al., 2010)) is dominant in the context of storing and processing big data, since it is highly available and scalable (Azzedin, 2013).

branches has its own configuration. In the Result Layer, the results are persisted, but not combined.

Kappa Architecture. To emulate the Kappa Architecture, BRAID's batch processing branch is deactivated. Incoming real-time data is stored in the Master Dataset and processed in the stream processing branch. Parallel processing jobs can be started at any time. The Master Dataset is used as a data source and configurations are stored in the Shared Storage.

Self-adjusting Mode. In addition to the aforementioned processing modes, BRAID enables to use and combine results from both branches within the processing logic for comprehensive analytics. For this, a data mining model is trained and tested in the batch processing branch using history data. This model can be stored in both, the Result Layer as well as the Shared Storage, where it is integrated into a configuration for the stream processing branch. Incoming real-time data can be classified using the model created from history data. By using VFDT, the model can be refined further by real-time data. This adjusted model is written back to the Shared Storage to be applied to both, history and real-time data. So, the application scenario introduced in Section 2 can be realized by using BRAID.

5 ASSESSMENT

In order to be suitable for comprehensive analyses, an architecture has to support all analytic steps described in Section 2. However, especially Step (5) (reuse of intermediate results for subsequent analysis steps) is not supported by any of the current state-of-the-art approaches.

Table 1 shows which processing steps are supported by which architecture. The Lambda Architecture enables both batch and stream processing, which is why the Steps (1) to (4) are supported. Although the results of both processings can be combined, the user has to do this manually by corresponding requests. Step (6) is therefore only partially supported. Step (5) is not realizable using the Lambda Architecture, as the Batch and Serving Layer and the Speed Layer are strictly separated. It is therefore not possible to automatically reuse results from the batch processing in stream processing. To realize this step, an administrator has to adjust the configuration for the batch or stream processing engine manually. The Kappa Architecture also enables batch and stream processing, realizing the Steps (1) to (4). Additionally, results can be combined easily, which supports Step (6). Yet, no insights gained from prior processing steps can be applied automatically to the processing logic for subsequent analyses as processing logic has to be defined statically beforehand. Therefore, Step (5) is also not supported by the Kappa Architecture. However, BRAID supports all six processing steps. As it is a hybrid architecture, all four batch and stream processing steps are supported. Like in the Kappa Architecture, a joint Result Layer enables the automatic combination of result sets (Step (6)). Additionally, the Shared Storage enables the reuse of results as, e. g., configurations. Therefore, BRAID is the only architecture which supports Step (5) and therefore all comprehensive analytics steps.

Table 1: Feature Support of Data Processing Architectures.

Step	*Lambda*	*Kappa*	*BRAID*
(1)	✓	✓	✓
(2)	✓	✓	✓
(3)	✓	✓	✓
(4)	✓	✓	✓
(5)	✗	✗	✓
(6)	(✓)	✓	✓

Thus, BRAID meets all requirements resulting from the use case given in Section 2. In the subsequent section, we discuss various implementation alternatives. That is, we assess the applicability of different processing engines for the implementation of BRAID.

6 IMPLEMENTATION CONSIDERATIONS

There are multiple implementation alternatives for both of BRAID's processing engines. This section discusses five of these data processing engines. Due to BRAID's modular structure, the engines can be replaced if necessary, based on a given optimization goal, e. g., runtime or memory usage. Similar to the Lambda Architecture, BRAID has two separated processing branches for batch processing and stream processing. Hence, the same processing engines as in the Lambda reference implementation can be used for BRAID:

Hadoop MapReduce[2] is a batch processing engine based on the *MapReduce* paradigm (Dean and Ghemawat, 2004) which is highly scalable and distributable. The data is processed in two separate steps: map and reduce. Input and output data are key-value pairs. In the map function, intermediate results are calculated from the data and forwarded to the reduce function as key-value-pairs. Here, data with the same key is consolidated.

[2] see https://hadoop.apache.org

Storm[3] is a stream processing system promoted as "Hadoop for real-time". Incoming real-time data is distributed to different nodes which process the data. If necessary, the data is repartitioned after each step.

However, since most processing engines for big data can process both, batches as well as data streams, for simplicity in terms of installation and maintenance effort, the same engine can be used for the batch and stream processing branch as well. Especially, the consolidation of BRAID's Shared Storage and the Result Layer is facilitated by using a common processing engine. Therefore, three engines are considered in the following, which can be used to realize the batch branch as well as the stream branch.

Spark[4] is highly fault tolerant alternative to MapReduce (Zaharia et al., 2010). Initially, it was a pure batch processing engine. Nowadays, it also supports stream processing using micro batches. Data arriving within a certain time window is computed as one micro batch. Additionally, Spark provides various extensions such as *Spark SQL* or *MLlib* for machine learning.

Samza[5] operates directly on data streams, similar to Storm. The stream is partitioned and distributed before being processed. To manage the distribution of the workload, Samza relies on *Hadoop YARN* (Vavilapalli et al., 2013). It uses *Kafka* (Kreps et al., 2011) to exchange messages between processing nodes. Samza offers an integration with HDFS but also provides an SQL interface as well.

Flink[6] is another stream processing engine. However, in contrast to Storm and Samza, Flink can process large-scale datasets as a kind of finite data streams. This approach ensures very low processing time. Additionally, Flink relies on continuous flows instead of micro batches. Similar to Spark, Flink provides many extensions, such as an SQL interface or *Flink ML* for machine learning.

Spark, Samza, and Flink support different data storage systems, such as HDFS or relational database systems. In contrast, Hadoop MapReduce depends on HDFS as it is a part of the Hadoop project. Due to this missing support of different data sources, it is less suited for BRAID. Samza is able to access various data storage systems through Kafka. Yet, it still needs the Hadoop framework due its usage of Hadoop YARN.

Spark and Flink can be flexibly applied to different applications due to their manifold extensions. These are the only two of the presented systems which enable both batch and stream processing out of the box. Other extensions enable the use of e. g., graph databases (Spark's *GraphX* or Flink's *Gelly*) or machine learning techniques (Spark's MLlib or Flink ML). In the context of BRAID, those extensions can be used to achieve a significant benefit. For instance, the machine learning libraries can be used to train machine learning models on the batch processing branch. These models can be applied to the processing of real-time data. Therefore, Spark and Flink are best suited for a BRAID prototype.

Since all of these approaches provide integration of HDFS, we also use HDFS for our implementation of the BRAID prototype. However, aside from HDFS, any other data storage systems and query languages can be used, such as relational database management systems (e. g., the *Oracle Database*[7]) or NoSQL approaches (e. g., *Google BigTable* (Chang et al., 2008)).

7 CONCLUSION

Due to the IoT, more and more contextual data are collected. A lot of knowledge can be gained from these data if they are analyzed comprehensively. This knowledge has a high economic value (e. g., to detect production problems at an early stage). To this end, a processing architecture for big data not only has to provide batch processing for history data and stream processing for real-time data, but also a tight coupling of these processing modes. That is, communication channels between the batch and stream processing engines are required in order to exchange data. Existing hybrid processing architectures for big data, namely the Lambda and the Kappa Architecture, lack such a tight coupling—here, batch and stream data are handled in two separate processes and have to be combined manually.

For this reason, we introduce BRAID, a novel hybrid processing architecture for big data, which intertwines the processing of history and real-time data by adding communication channels between the batch and stream engine. Via these channels, not only data is exchanged, but also the configuration of the batch or stream engine can be controlled. That is, the processing logic can be modified automatically at runtime, whereby BRAID is able to react to altered conditions. We discuss different implementation techniques in order to find a suited implementation strategy for BRAID. Evaluation results show that BRAID enables more flexible and accurate data analytics than the Lambda and Kappa Architecture.

In the IoT domain, privacy is a critical issue concerning the processing of data. Future work has to consider, how to realize privacy protection in BRAID

[3] see https://storm.apache.org
[4] see https://spark.apache.org
[5] see https://samza.apache.org
[6] see https://flink.apache.org
[7] see https://www.oracle.com/database/index.html

without compromising analytic quality. A possible approach towards this goal is PATRON (Stach et al., 2017, Stach et al., 2018).

ACKNOWLEDGEMENTS

This paper is part of the PATRON research project which is commissioned by the Baden-Württemberg Stiftung gGmbH. The authors would like to thank the BW-Stiftung for financing this research.

REFERENCES

Abadi, D. J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., and Zdonik, S. (2003). Aurora: A New Model and Architecture for Data Stream Management. *The VLDB Journal — The International Journal on Very Large Data Bases*, 12(2):120–139.

Azzedin, F. (2013). Towards a Scalable HDFS Architecture. In *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems*, CTS '13, pages 155–161.

Barbierato, E., Gribaudo, M., and Iacono, M. (2013). Modeling Apache Hive Based Applications in Big Data Architectures. In *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, ValueTools '13, pages 30–38.

Beheshti, A., Benatallah, B., Nouri, R., Chhieng, V. M., Xiong, H., and Zhao, X. (2017). CoreDB: A Data Lake Service. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 2451–2454.

Casado, R. and Younas, M. (2015). Emerging Trends and Technologies in Big Data Processing. *Concurrency and Computation: Practice and Experience*, 27(8):2078–2091.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2008). Bigtable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems*, 26(2):4:1–4:26.

Columbus, L. (2016). Industry 4.0 Is Enabling A New Era Of Manufacturing Intelligence And Analytics. *Forbes*. https://www.forbes.com/sites/louiscolumbus/2016/08/07/industry-4-0-is-enabling-a-new-era-of-manufacturing-intelligence-and-analytics.

Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation – Volume 6*, OSDI '04, pages 137–149.

Domingos, P. and Hulten, G. (2000). Mining High-Speed Data Streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80.

Geissbauer, R., Schrauf, S., Koch, V., and Kuge, S. (2014). *Industry 4.0 – Opportunities and Challenges of the Industrial Internet*. PricewaterhouseCoopers.

Gröger, C. (2018). Building an Industry 4.0 Analytics Platform. *Datenbank-Spektrum*, 18(1):5–14.

Gröger, C., Schwarz, H., and Mitschang, B. (2014). Prescriptive Analytics for Recommendation-Based Business Process Optimization. In *Proceedings of the 17th International Conference on Business Information Systems*, BIS '14, pages 25–37.

Kreps, J. (2014). Questioning the Lambda Architecture. *O'Reilly Media*. https://www.oreilly.com/ideas/questioning-the-lambda-architecture.

Kreps, J., Narkhede, N., and Rao, J. (2011). Kafka: A Distributed Messaging System for Log Processing. In *Proceedings of the 6th International Workshop on Networking Meets Databases*, NetDB '11, pages 1–7.

Marz, N. (2011). How to beat the CAP theorem. http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html.

Marz, N. and Warren, J. (2015). *Big Data - Principles and best practices of scalable real-time data systems*. Manning Publications Co.

Middleton, P., Kjeldsen, P., and Tully, J. (2013). Forecast: The Internet of Things, Worldwide. *Gartner, Inc.* http://www.gartner.com/document/2625419.

Quix, C., Hai, R., and Vatov, I. (2016). Metadata Extraction and Management in Data Lakes With GEMMS. *Complex Systems Informatics and Modeling Quarterly*, 9(9):67–83.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The Hadoop Distributed File System. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10.

Stach, C., Dürr, F., Mindermann, K., Palanisamy, S. M., Tariq, M. A., Mitschang, B., and Wagner, S. (2017). PATRON – Datenschutz in Datenstromverarbeitungssystemen. In *Tagungsband der 47. Jahrestagung der Gesellschaft für Informatik e.V.*, pages 1085–1096. *(in German)*.

Stach, C., Dürr, F., Mindermann, K., Palanisamy, S. M., and Wagner, S. (2018). How a Pattern-based Privacy System Contributes to Improve Context Recognition. In *Proceedings of the 2018 IEEE International Conference on Pervasive Computing and Communications Workshops*, CoMoRea '18, pages 238–243.

Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., O'Malley, O., Radia, S., Reed, B., and Baldeschwieler, E. (2013). Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster Computing with Working Sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud '10, pages 1–7.