EasyChair Preprint

№ 1336

Diting: A Real-time Distributed Feature Serving System for Machine Learning

Meng Wan, Chongxin Deng and Siyu Yu

July 28, 2019

Diting: A Real-time Distributed Feature Serving System for Machine Learning

Meng Wan, Chongxin Deng, Siyu Yu
Vectorlab, JD Finance,
Beijing 100176, China,
wanmeng@jd.com, dengchongxin@jd.com, yusiyu@jd.

***Abstract*—4G networks dramatically boost the speeds and coverage of networks, and there are mounting mobile data produced by data sources of 4G. The generated data can be applied into preventing financial criminal activities, for example, account leakage risks can be prevented with such data instantly, which demands short time delay to predict the fraud, at most within 50 ms. Furthermore, the lower latency, higher data capacity of forthcoming 5G networks will create a new platform for the delivery of services wherever the 5G network exists, paving a way for artificial intelligence-based banking services. Real-time prediction services and data analytics services that can be integrated into business applications have become eagerly demanded in recent years. However, most machine learning and deep learning platforms only provide offline model training&test and model predictions without real-time feature processing. Diting is an easy-to-use distributed intelligent machine learning platform, providing offline model training and low-latency feature serving for predictions in real time. Diting offers incremental feature computing, combines technologies of resource scheduling, rule engines, and Remote Procedure Call (RPC), and builds a real-time distributed computing framework, thus offers low-latency end-to-end prediction services. Diting is also a collaborative, drag-and-drop and virtualization tool. Domain expert users without any programming knowledge can quickly fulfill their business logic. Using real production data for over two months, we show that Diting tremendously improves productivity, and bridges the gap between offline and real-time feature engineering.**

Keywords: Real-time Computing, Incremental Aggregation, Distributed Systems, Feature Engineering.

I. Introduction

Since the first 4G networks were launched, e-commerce has undergone an extensive proliferation in the past decade, now over 80% of online shopping is running on 4G networks in Jingdong (JD), the world's third-largest internet company by revenue and China's largest online retailer. JD Finance, the wealth management platform of JD, was established to give individuals and businesses quick, easy and convenient access to the financial service. The pervasion of 5G networks could revolutionize the e-commerce and the finance industry by Virtual Reality (VR), Augmented Reality (AR) technologies and predictive machine learning-based services. These services generally collect a user's behavioral data in real time to recommend location-based financial advice, such as suggesting new ways to save at the retail store and offering more precise and valuable wealth management consultation.

Nowadays, machine learning (ML) has come to play an integral role in many stages of the financial ecosystem, from assessing risks to granting loans, to the handling of bad assets, to fraudster detection. ML models are required to be trained with a wide range of features. Solving a ML problem often involves the following steps: gathering data, cleaning data (ETL), feature engineering (FE), model training, testing and prediction [1]. ETL stands for Extract, Transform, Load, and it aims to provide clean data before FE. In Diting, ETL is incorporated into FE. In order to build a model that can make the best prediction or recommendation, the first thing one needs to do is to generate appropriate features from the datasets; better features are the deciding factor of the performance of a system.

With advancements in technology, more and more predictions are expected to be done in real time [2]. Identifying fraudulent purchases in a particular time frame in e-commerce scenarios can prevent losses of millions of dollars per year. However, FE of traditional artificial intelligence platforms often focused on a batch or offline FE. It would take a team of data engineers additional months to develop a manual pipeline of real-time FE. To the best of our knowledge, there is no precise report concerning systems that automatically provide feature serving in real time using the same logic as offline feature cleansing. *Real-time feature serving* is defined as providing the feature processing service for real-time ML prediction, bringing feature data over the network in real time.

Rather than force the session to process the entire source data from scratch, incremental computing is generally adopted to do the real-time computing. Intuitively, incremental FE computation can be implemented using *dynamic/incremental algorithms* methods, where programmers need to design the logic that maintains the value of a function over an evolving set, for example, tracking the maximum amount of payment that changes with insertions or deletions of incoming tuples [3]. Nevertheless, after a large body of prior research in the incremental algorithms, we found that although they are efficient, they are usually complex to realize even for simple *fully dynamic* questions like dynamic connectivity [4], [5], [6]. Therefore, we resort to *incremental sliding window analytics* [6] to tackle real-time computation, and better yet, certain characteristics of sliding windows are exploited so that the time consumption of a single incoming tuple is not dependent on the size of the whole historical data but rather commensurate with the number of intermediate results.

In summary, the main contributions of this paper are ab-

stracted as follows:

- **an easy-to-use distributed ML framework and analytics:** Machine learning can be hard to grasp for most people, especially those who are from a non-programming background. Diting is a drag-and-drop tool that automates machine learning and does not require any coding. It's a visualized modeling tool that lets users build custom models, train them, tune them by adjusting parameters, and deploy them in applications. The model is built in the distributed cluster, its performance is guaranteed.
- **support for custom-built end-to-end FE pipelines:** Diting eliminates most of the draining operations related to preparing features for ML, ranging from cleansing, normalization, feature extraction to feature transformation.
- **offer a unified framework for real-time and offline FE and model prediction service:** FE is the process of constructing new features from existing data to train an ML model. Diting removes human-labor processes for users, provides an automatic offline-to-real-time projection of FE & ML pipelines.

II. RELATED WORK

Spark streaming [7] processes data streams in mini-batches, where each mini-batch collects a set of events that arrived over the batch period. The minimum interval at which data received is recommended to be 50 ms. While we require each and every record is processed as it arrives, and the maximum latency is 50 ms.

Storm uses topologies to do real-time computation. For a Storm topology, the user has to specify the amount of resources, and the resources is not shared between topologies, therefore the maximum amount of runnable topologies is finite. Whereas in Diting, real-time DAGs can be dynamically added or removed as rules, are evenly distributed in the cluster by dynamic scheduling, so the resources of CPU and memory of the cluster isn't occupied exclusively by any DAG, there is no limitation of the number of the DAGs. Once the average loads of CPUs or the heap size per worker (JVM) is too high on a cluster, new servers can be added to the cluster. Besides, Storm still requires manual programming, whilst coding is removed from Diting system by configuring rules.

III. IMPLEMENTATION & DESIGN

Since the problem of *Real-time feature serving* is generic, any other organizations can draw on the experience we describe here no matter what industries they involve in as long as unifying offline and real-time FE is business-critical.

A. Highlights of Our Approach

Incremental Aggregation for FE. Incremental aggregation aims to tackle the problem of real-time computing of a limited set of aggregate operators such as $mean$, max, min, sum, $count$, .etc, and a large number of complex aggregates of the above basic operators. Besides, one predictive computation is customized for certain business scenarios. If a customer demands more features, then accordingly the pipelines could be adjusted to it.

We first consider some fundamental operators in database systems and Data Stream Management Systems (DSMS) [8], [9], [10], [11]. They are categorized into *distributive* (e.g., max, min, sum, and count) aggregates and *algebraic* (e.g., avg, std, trend, skew) aggregates, and some *holistic* aggregates (e.g., median, mode), organized by [12].

The incremental aggregates we support are listed as follows: mean, standard deviation, sum, min, max, count, distinct count, trend, time since last time, last value, skew, and feature generation derived from previous aggregates.

Considering continuous tuples from 4G networks are deemed as inputs of downstream of the streaming system like Apache Storm [13], the featre serving automatically will trigger an incremental aggregation pipeline to compute the statistics instantly, without recomputing on a large chunk of historical raw data from scratch. These results will be stored in a fast key-value database, such as Redis, to offer a prompt fetch for the following predictive service.

Two-Phases FE processing algorithm: We create a *Two-Phases FE processing algorithm*. In phase A, the pipeline precomputes all the incremental intermediates for the required measures; in phase B, the pipeline doesn't do any incremental calculation and directly fetches the intermediates produced in phase A and make a simple calculation to derive the final results. The motivation of separating the two phases is to achieve the real-time outcome of the features and minimize the computation and I/O transmission. The phase A takes place right after a new tuple arrives at a window and propagates the update of the tuple with the previous aggregated value. The phase B relies on a few intermediates, which means it is in the blink of an eye. The prerequisite of this approach is that we know in advance about the sort of aggregates and features that are needed in an application. The customization characteristic of the framework is adopted to improve the performance of the response time. For example, $average$ can be treated as $\frac{sum}{count}$. sum and $count$ are incrementally calculated in the phase A ahead, and The phase B only merges the incremental intermediate and the value of the current tuple, and does a division of sum and $count$, and similarly standard deviation relies on *sum of squares, sum, count*.

Rule-based Real-time Distributed Computing The starting point of a model training&test and model prediction service is the generation of a *user directed acyclic graph* (DAG, also known as workflow), composed of nodes produced by our user in the Web UI. The DAG will be automatically mapped into two separate versions, an offline DAG and an real-time DAG. Offline DAGs are different from real-time DAGs because real-time DAGs do not include all the nodes in offline DAGs, such as model training or test, model evaluation, etc. In a DAG, a node represents a task, which can be mainly classified into two categories, one is feature related, another is model related; and an edge stands for the data dependencies between tasks, which makes the nodes connected in a pipelined manner. Through a few clicks of drag-and-drop and parameter

configuration, a user of ML scientist can model a request in a DAG like Fig. 1.

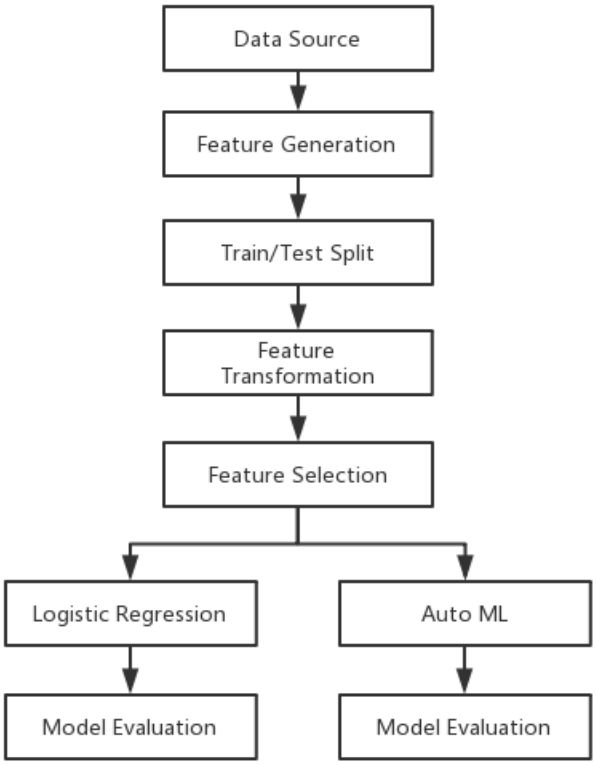

Fig. 1: DAG example

It is worth noting that multiple DAGs could be executed because multiple users could be using the system simultaneously, and a user may create any number of different DAGs at any time, and after model training and test, a user can deploy a DAG service. All DAGs can be executed parallel simultaneously. For offline DAGs, there are many mature resource management and job scheduling systems could help schedule jobs of offline DAGs, such as Hadoop Yarn [14], Apache Mesos [15]. For real-time DAGs, a distributed computing framework is required for adding and removing real-time DAGs dynamically.

The benefit of the rule-based approach is that users only need to draw a user DAG and configure parameters, no need of writing codes. The details will be discussed in Chapter 3.

B. Incremental Windowing

Windows are at the core of processing streams [8], [16], in this paper, we employ an *incremental update* mechanism that can embody incremental changes of the newly arrived tuple and update the intermediates of the measure attributes.

Among the three types of sliding windows: Hopping, Tumbling, and Overlapping, we focus on *Tumbling windows* (also called Fixed windows). Tumbling windows are defined by a static window size, and the window size equals the sliding period [17], [18]. The tumbling window processes queries in a non-overlapping manner. The only difference between our tumbling window and traditional tumbling window is that traditional tumbling window waits until window time is over, and process only once when all the tuples belong to the window is collected. We create an incremental tubmling window, once an incoming tuple arrives, it will get processed at once, so it does not cover the entire input when a windows starts. The current window tumbles on a window-size basis. when the last window time is over, and the window is empty. As time goes by, the input tuples arrives, the window gets filled in real time until it is full. Therefore, we achieve better real time performance by not waiting for the whole length of a window size. For example, a one-minute tumbling window incrementally processes tuples that occurred in one minute and slides after the minute passed.

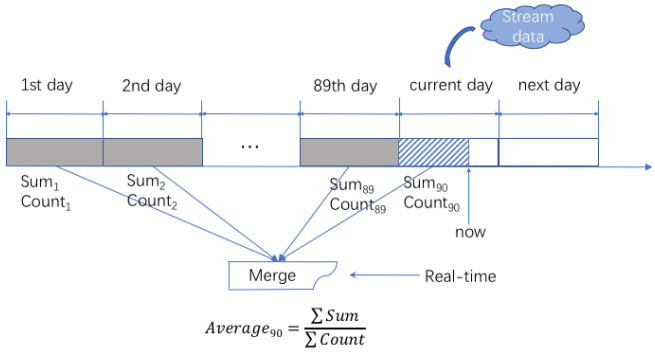

Fig. 2: Windowing design and merge, with an example of computing the average of 90 days.

Windowing chops up an unbounded stream into finite dataset based on time or tuples, and then process each window as a separate group [19]. We support four types of windows sizes, one day (daily), one hour (hourly), one minute, and one second. The upper limits of caching number for each window are 180 days, 24 hours, 60 minutes, and 60 seconds, and we require features within the longest time span of 180 days be available in 50 ms at most. These various window sizes are fairly enough for generating features for possible user queries we collected.

The tumbling window offers an advantage of simplicity for later window merging that joins integer number of windows. The window merging logic subjects to both the window partitioning and sliding methods, it regards the current day as an integer day no matter whether it has finished, then glues the remanent windows together. If a query is about to fetch a feature in the past 90 days, a combination output of eighty-nine-days windows and one incomplete accumulating window will be returned as shown in Fig. 2.

IV. Rule-based Real-time Distributed Computing

It is hard to let users do custom coding and implement the model training and serving especially when requirements change frequently. With the combination of real-time technologies of *resource scheduling, rule engine, RPC HA service*,

the rule-based real-time distributed computing framework is established.

Resource scheduling involves allocating system resources according to the computational load of given tasks, which is a crucial technique for providing high performance distributed computing [20], [21]. Scheduling of tasks in distributed computing systems aims to schedule when and where each task should be executed in a cluster. As can be seen in Fig. 3, the feature service master does the scheduling by checking which worker has the least running jobs.

Rule engines [22] are a mechanism for executing "business rules", interprets complicated condition/action statements, often in the form of "if/then" notations. Rules are stored in a rule database, then the engine watches over the input stream, matches the data against the existing rules in real time, and then execute appropriate actions when conditions are met and therefore can keep data moving at low latency [22]. The biggest advantage of rule engines is that they decouple business logic and application code, which means business logic doesn't need to be hard-coded into the program with procedural languages. All the business logic is centralized, so it is easier to maintain, enhance and update new logic, which is a pluggable component that doesn't have to restart the system for rule changes or deploying new executable rule.

In rule engines, we make one rule represent one DAG executor or one Node executor. A worker and DAGs are of one-to-many relationship. Assume a rule is a DAG, when building a rule engine, and deploying a new DAG to a worker, to decide on which worker the DAG should be deployed, the resource scheduler of the master judges on-the-fly which worker is running the least number of DAGs, then allocates the DAG to the worker. So the association between a DAG and a worker is a one-to-one mapping.

RPC is a protocol that allows one program to request a service of a program located in another computer on a network as if it were a normal local program. We use a cross-language RPC framework as a fundamental technique for distributed services. It provides high availability (HA) in the production environment and supports load-balance. We develop the distributed framework by designing a master and many workers on the cluster. Each worker or master is redundantly distributed in the cluster , if any worker or master failed, the system still provides normal service. The number of workers on a RPC service lies in the CPU and main memory capacity of it. The logical structure of rule engines and distributed services is illustrated in Fig. 3. A job can be a feature serving node or a real-time DAG executor, the significant characteristic of the real-time distributed computing architecture is that both of feature serving and DAG executors are working on the mechanism of it.

When a message arrives, each message has a "topic" attribute, representing the data source of DAGs, and many DAGs may have the same topic (data source), so the master looks up the topic and gets which DAG the message belongs to. Since a DAG and a worker are of one-to-one relation, the master matches the topic with a worker, then dispatches the message

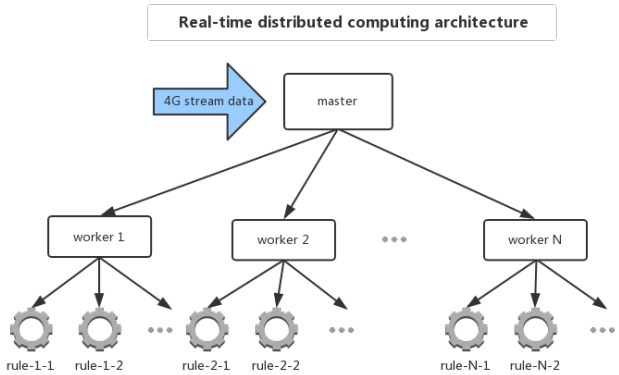

Fig. 3: Real-time distributed computing architecture. If a message from feature serving or a real-time DAG is matched with a rule, a job will be started as a rule executor.

to all the workers with related DAGs, and execute the DAGs concurrently.

Fig. 4 shows how users of Diting interact with the system and work in order. From the user perspective, he or she by drag-and-drop can create an application model as a user DAG, After one click of the "Save" button, the offline system will generate an offline DAG transparently in the background. Then the user click "Train" button, the offline system will run the offline DAG which executes feature engineering and model training and test altogether.

Now, users can click "Deploy Prediction Service" button to add related rules to the real-time service, it will convert the user DAG to a real-time DAG; finally, the user click "Start Prediction Service" button, the system will add the real-time DAG rule to rule-based real-time distributed computing service, and the rule of the service takes effect . The user is free of coding throughout the process.

V. EXPERIMENTAL EVALUATION

We built Diting system on a cluster of 10 servers, each with hardware as follows: 2 x Intel ® Xeon ® E5-2650 v4 of CPU, 16 x DDR4 2133Hz 16GB of RAM, 8 x SAS 2.5" 15K 6Gb 600GB (RAID 10) of storage, 2 x Intel X520 10Gb of network.

We present an experiment of a login model to show the performance of Diting. Login model is an approach to capture outliers of login behaviors in JD over 220 million active users, it captures the sequence of login actions, uses the model of LightGBM to predict the probability of a user account being hacked and at risk of capital losses. The LightGBM is a gradient boosting framework that uses tree based learning algorithms [23].

We evaluate the performance of Diting by doing feature computing for the login model on the real production streaming data, the features calculated by Diting is mainly about the count of login behaviors in the past 2/5/24 hours, and average time of a login action, the distinct count of located cities

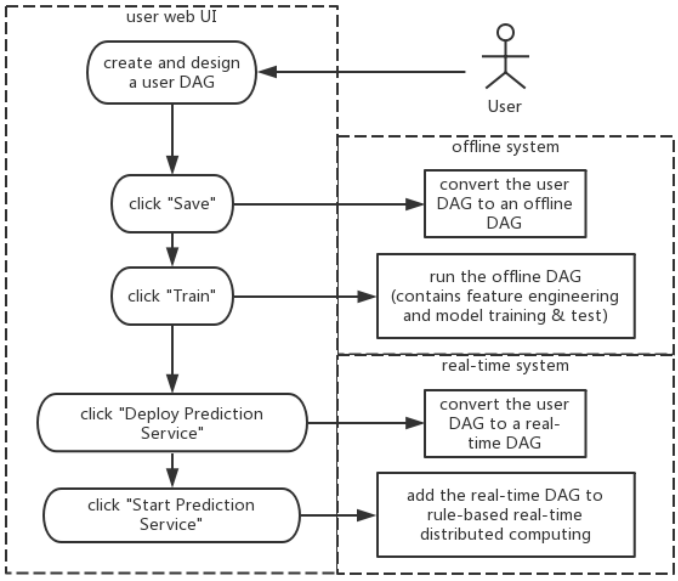

Fig. 4: Diting system flowchart. Users enjoy the real-time end-to-end feature serving and prediction service.

of a login action, etc. There are about 27 features produced in a feature engineering. Fig. 5 shows the prediction service latency of a tuple is done on a dataset of two months, the average time of a feature computing is circa 4.1 ms, which fills the lacunae between offline and real-time feature engineering, and tremendously improves productivity. Traditional methods of developing a pipeline of feature serving take about two months. 100% accuracy is guaranteed as long as the offline and real-time processing are working on the same data.

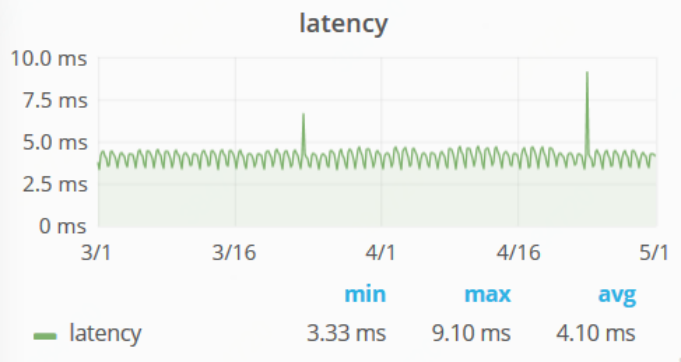

Fig. 5: Average latency running on a real production streaming data source for two months. The two pulses of the figure are due to network instability.

VI. CONCLUSIONS

We presented Diting, a real-time feature serving system with an easy-to-use interface, and a rule-based distributed computing framework. We showed that Diting can create millions of features both in real-time and offline circumstances. We also validated that Diting is real-time in a production cluster with

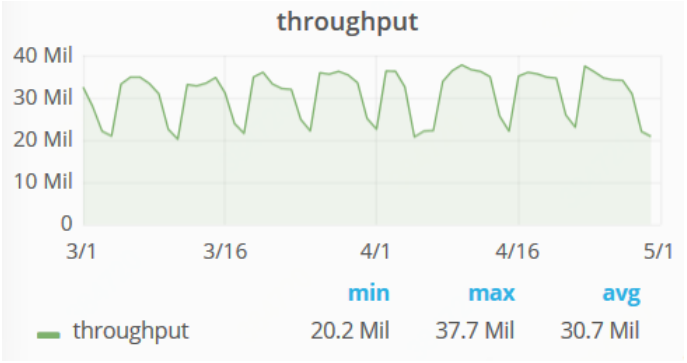

Fig. 6: Throughput aggregated by day running on a streaming data source for two months.

inspiring results. Incremental aggregates of more basic features and more accurate feature computing are our future work.

REFERENCES

[1] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[2] P. Yang, S. Thiagarajan, and J. Lin, "Robust, scalable, real-time event time series aggregation at twitter," in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018, pp. 595–599.

[3] Y.-J. Chiang and R. Tamassia, "Dynamic algorithms in computational geometry," *Proceedings of the IEEE*, vol. 80, no. 9, pp. 1412–1434, 1992.

[4] D. Alberts, *Implementation of the dynamic connectivity algorithm by Monika Rauch Henzinger and Valerie King*. Citeseer, 1995.

[5] B. M. Kapron, V. King, and B. Mountjoy, "Dynamic graph connectivity in polylogarithmic worst case time," in *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2013, pp. 1131–1142.

[6] P. Bhatotia, U. A. Acar, F. P. Junqueira, and R. Rodrigues, "Slider: incremental sliding window analytics," in *Proceedings of the 15th International Middleware Conference*. ACM, 2014, pp. 61–72.

[7] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters." *HotCloud*, vol. 12, pp. 10–10, 2012.

[8] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," *Vldb Journal*, vol. 12, no. 2, pp. 120–139, 2003.

[9] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina *et al.*, "The design of the borealis stream processing engine." in *Cidr*, vol. 5, no. 2005, 2005, pp. 277–289.

[10] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, "Millwheel: fault-tolerant stream processing at internet scale," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1033–1044, 2013.

[11] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 147–156.

[12] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, "Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals," *Data mining and knowledge discovery*, vol. 1, no. 1, pp. 29–53, 1997.

[13] D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications." in *OSDI*, vol. 10, 2010, pp. 1–15.

[14] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[15] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *NSDI*, vol. 11, no. 2011, 2011, pp. 22–22.

[16] P. Carbone, J. Traub, A. Katsifodimos, S. Haridi, and V. Markl, "Cutty: Aggregate sharing for user-defined windows," in *ACM International on Conference on Information and Knowledge Management*, 2016, pp. 1201–1210.

[17] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt *et al.*, "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015.

[18] S. Krishnamurthy, C. Wu, and M. Franklin, "On-the-fly sharing for streamed aggregation," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, 2006, pp. 623–634.

[19] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker, "Semantics and evaluation techniques for window aggregates in data streams," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM, 2005, pp. 311–322.

[20] M. Kalra and S. Singh, "A review of metaheuristic scheduling techniques in cloud computing," *Egyptian informatics journal*, vol. 16, no. 3, pp. 275–295, 2015.

[21] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, p. 63, 2015.

[22] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM Sigmod Record*, vol. 34, no. 4, pp. 42–47, 2005.

[23] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, 2017, pp. 3146–3154.