SPSC: Stream Processing Framework Atop Serverless Computing for Industrial Big Data

Zinuo Cai, Zebin Chen, Xinglei Chen, Ruhui Ma, *Member, IEEE*, Haibing Guan, *Member, IEEE*, and Rajkumar Buyya, *Fellow, IEEE*

***Abstract*—With the advance of smart manufacturing and information technologies, the volume of data to process is increasing accordingly. Current solutions for big data processing resort to distributed stream processing systems, such as Apache Flink and Spark. However, such frameworks face challenges of resource underutilization and high latency in big data application scenarios. In this article, we propose SPSC, a serverless-based stream computing framework where events are discretized into the atomic stream and stateless Lambda functions are taken as context-irrelevant operators, achieving task parallelism and inherent data parallelism in processing. Also, we implement a prototype of the framework on Amazon Web service (AWS) using AWS Lambda, AWS simple queue service, and AWS DynamoDB. The evaluation shows that compared with Alibaba's real-time computing Flink version, SPSC outperforms by 10.12% when the overhead is close.**

***Index Terms*—Big data, cloud computing, intelligent industry, serverless computing, stream processing.**

I. Introduction

NOWADAYS, large amounts of data are generated to process as the development of information and communication technologies, such as the Internet of Things (IoT), industrial sensors, sensor networks, etc., which impact manufacturing profoundly. With the technologies, data generated from modern manufacturing systems are experiencing explosive growth, which has reached over 100 EB annually [1]. The manufacturing data contains rich knowledge, driving the transformation of the conventional manufacturing paradigm to the intelligent manufacturing paradigm. Smart manufacturing [2] utilizes the concepts of cyber–physical systems with the IoT, cloud computing, service-oriented computing, artificial intelligence [3], [4] and data science, which would be the hallmark of the next industrial revolution [5].

To deal with big data processing tasks, researchers resort to distributed stream processing systems where data is continuously generated as streams and processed by distributed and low-latency computational frameworks. The typical distributed data stream processing frameworks include open-source frameworks, such as Storm, Spark Streaming, Flink, and Kafka Streams, and proprietary frameworks, such as IBM Streams. Researchers have proposed combining cloud and edge computing with stream processing systems to process the data streams with short delays and deal with the large volume of data. Apache Flink and Spark are general-purpose streaming data processing frameworks.

However, existing methods are not suitable for the increasing amount of data. On the one hand, an abundance of computing infrastructure and resources remain underused with the increasing growth of the IoT and edge computing. On the other hand, due to the latency issues and networking overhead, today's cloud models suffer from high latency and response time when processing large volumes and varieties of data. The responsibilities of managing underlying infrastructure are heavy for developers, and the process is mainly manual, task-specific, and error-prone [6].

Fortunately, we identify that serverless computing [7], [8], [9], an emerging cloud computing paradigm, can effectively solve the difficulties faced by cloud computing when applied to collecting and processing industrial big data. Serverless computing does not mean that there are no servers in the cloud. Still, the operations of servers, such as application and release, and scaling up and down, are handled by cloud vendors other than users. Serverless computing is also known as function as a service (FaaS) because cloud computing users only need to write code and complete the running logic of the application. Cloud computing providers should meet other requirements for code operation, such as lightweight virtualization, to execute functions, external databases to save the status of functions, and monitoring and log services. These backend services are the key to ensuring the safe and correct execution of stateless function instances. Therefore, serverless computing is featured as FaaS for cloud computing users and backend-as-a-service (BaaS) for providers.

Introducing serverless computing to stream industrial data processing solves the two challenges mentioned above. First, cloud vendors can effectively improve the resource utilization efficiency of their underlying hardware. Serverless computing

Manuscript received 31 January 2023; revised 2 May 2023 and 28 July 2023; accepted 13 August 2023. Date of publication 21 June 2024; date of current version 30 October 2024. This work was supported in part by the Shanghai Key Laboratory of Scalable Computing and Systems, and in part by the Eighth Research Institute of China Aerospace Science and Technology Group Company, Ltd. under Grant USCAST2023-17 and Grant USCAST2023-21. This article was recommended by Associate Editor Y. Tang. *(Corresponding author: Ruhui Ma.)*

Zinuo Cai, Zebin Chen, Xinglei Chen, Ruhui Ma, and Haibing Guan are with the Computer Science and Engineering Department, Shanghai Jiao Tong University-Minhang Campus, Shanghai 200240, China (e-mail: kingczn1314@sjtu.edu.cn; czb453874483@sjtu.edu.cn; chen1556805501@sjtu.edu.cn; ruhuima@sjtu.edu.cn; hbguan@sjtu.edu.cn).

Rajkumar Buyya is with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, The University of Melbourne, Melbourne, VIC 3052, Australia (e-mail: rbuyya@unimelb.edu.au).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCYB.2024.3407886.

Digital Object Identifier 10.1109/TCYB.2024.3407886

can automatically scale up and down according to the request arrival rate, meanwhile the traditional cloud computing model, which requires reserving sufficient computing resources to cope with the surge of industrial data. Moreover, because serverless computing takes functions as the primary resource application unit, lightweight virtualization is more agile than virtualization schemes represented by virtual machines, reducing the execution delay of the stream processing system.

Therefore, we propose SPSC, a stream computing framework to handle industrial data atop serverless computing platforms in this article. We conceptually divide the events of the processing process into several subsets and call them atoms. The lowest level of operation of the framework is the atomic level, and each computing unit is also designed to perform atomic processing. In other words, the framework workflow is a computational diagram of implementing transactional microservices on the atomic flow using data flow semantics. Therefore, Lambda functions become operators in stream computing, and users only need to pay attention to atomic-level transaction business logic when coding. Then, we use Amazon Web service (AWS) simple queue service (SQS) as a message queue to realize the communication between operators and the storage of intermediate states. The visibility of SQS ensures the At Least Once mechanism of the framework. We also use AWS DynamoDB as the framework's persistent storage solution for state storage so that users can quickly expand the database and do not need to design data relationships. We believe that the proposal of SPSC can provide a feasible and cost-effective way of thinking for Big data processing in industry, and further improve the feedback from data processing to production.

In this article, we have made the following contributions.

1) We propose a stream computing framework atop serverless architecture. Our approach combines the fundamental idea of the stateful data flow model with the serverless architecture, divides the event into atoms, and then uses the stateless Lambda function to realize the operator in the stream computing. We also use AWS SQS and AWS DynamoDB as message queues and persistent storage solutions.
2) Based on the designed framework, we propose a parallel computing method. Operators are context-irrelevant by discretizing the processing process into an atomic stream, thus achieving task parallelism. The automatic adjustment of the concurrency of Lambda instances and the polling mechanism of message queues make data parallelism an inherent framework attribute.
3) We have implemented a prototype of the framework and evaluated the performance of our implementation. According to our experimental results, compared with Alibaba's real-time computing Flink version, which is charged according to the lease duration, our framework can save 10.8% of the cost on average under the same computing tasks. Our framework can improve the performance by 10.12% on average when the overhead is close.

In Section II, we provide a detailed introduction to the background knowledge and motivation related to serverless and data streaming processing. In Section III, we introduced the design concept and some important features of SPSC. In Section IV, we compared SPSC with Alibaba Flink and concluded that SPSC would be more cost effective when its performance was close. In Section V, we introduce research on serverless applications and data processing.

II. Background and Motivation

A. Serverless Computing

Serverless computing is a new paradigm of cloud computing. Its typical feature is FaaS, which is evolved from infrastructure-as-a-service (IaaS) [10] and platform-as-a-service (PaaS) [11]. Compared with the traditional cloud computing paradigm, serverless computing has the following advantages in the processing and collecting industrial big data. First, serverless computing provides more cost-efficient cloud services because of its "pay as you use" billing mode. Users only need to pay for how many cloud computing resources they use. On the contrary, in the PaaS mode, users need to charge for the occupied resources, even if the resources are not used at all. Second, serverless computing provides an excellent ability to automatically scale up and down, effectively responding to the change and frequency of industrial data volumes. Because the generation of industrial data is a process of time series change, it is necessary to automatically increase or decrease the function instances according to the amount of data. Because serverless computing provides a lightweight application execution environment, container creation or destruction is faster than virtual machines. Third, serverless computing can reduce the time and experience of operation and maintenance personnel. Serverless computing is not only featured as function-as-a-service but also as BaaS, including object storage, load balancing, resource scheduling, and other backend services, to meet the needs of serverless computing.

Fig. 1(a) shows the typical framework design of a serverless computing platform. Among the components of a serverless platform, Gateway is usually the platform's entrance responsible for receiving user requests. Controller is usually the core of the framework and undertakes the functions of user request processing, load balancing, function instance creation and destruction, etc. Executor is the execution environment of functions, usually lightweight containers or virtual machines, such as Docker, gVison, or Firecracker [12]. Open-source serverless computing platforms include OpenWhisk [13], OpenFaas [14] and Fission [15]. Fig. 1(b) shows the framework of OpenWhisk. Its Gateway uses the open-source reverse proxy server NGINX, and the Pod is composed of a Docker container in the Kubernetes cluster corresponding to the executor. Commercial serverless computing frameworks include Amazon's AWS Lambda, Google Cloud and Alibaba Cloud's Function Compute. The architecture of AWS Lambda is shown in Fig. 1(c). After the frontend receives the user's request, the worker manager will schedule and start the function instance. The function metadata will be sent to the corresponding worker for execution. Our subsequent work will be based on AWS Lambda.

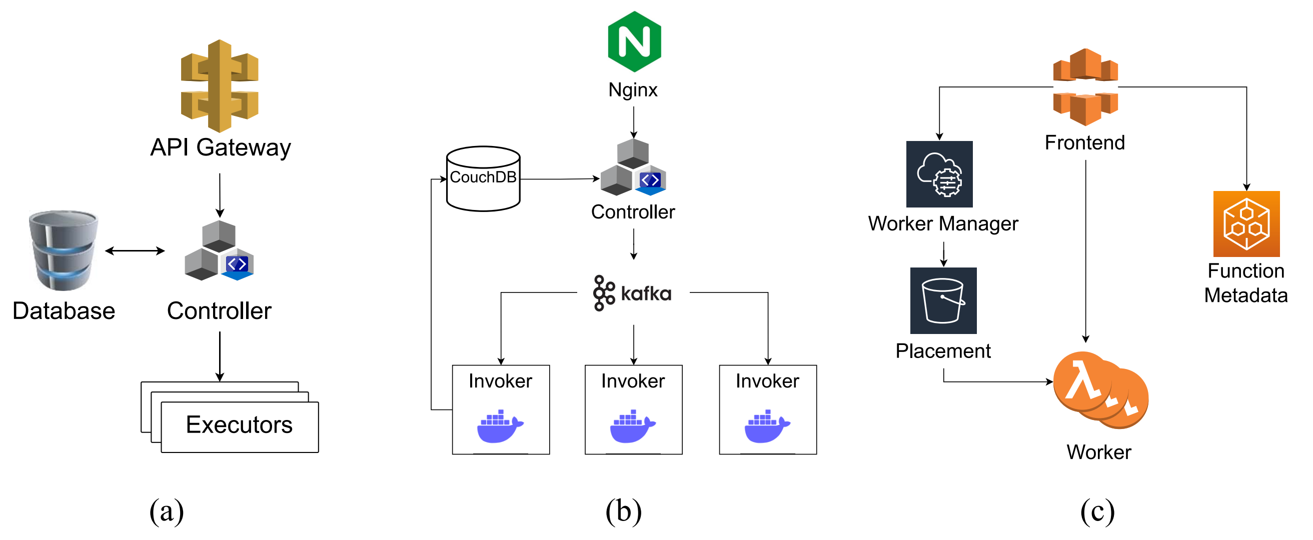


Fig. 1. Architectures of serverless platforms. (a) Typical serverless framework. (b) OpenWhisk. (c) AWS lambda.

B. Data Stream Processing

Stream processing is the processing of data which are produced in a stream of events. Unlike traditional batch processing, where static data are stored in a database, a file system, or other forms of mass storage and handled as needed, stream processing processes dynamic or continuous data as an event upon receiving one from the stream. A stream is an unbounded sequence of events generated continuously in time from the source to the sink. Stream processing pipelines often involve multiple operations, such as filters, aggregations, analytics, transformations, enrichment, branching, joining, etc. As unbounded and global datasets are increasingly becoming common and essential in day-to-day business [16], most data are born as continuous streams, such as sensor measurements, Weblogs, mobile usage statistics, and financial trades. The stream processing market is experiencing exponential growth with applications relying heavily on real-time analytics, inferencing, and monitoring, such as telecommunications, smart cities, health care, transportation [17], retail, manufacturing, advertising, cyber security [18], and finance.

Data processing systems are evolving to be more stream-oriented, where each data record is continuously processed as it arrives by distributed and low-latency computational frameworks. Currently, multiple distributed data stream processing frameworks, open source (Storm, Spark Streaming, Flink, and Kafka Streams) and commercial (IBM Streams), exist for ingesting, processing, storing, indexing, and managing streaming data. Research on data stream processing engines has diverged into four directions: 1) query-based systems, such as NiagaraCQ [19], TelegraphCQ [20], and AsterixDB [21]; 2) online distributed machine learning systems, such as scalable advanced massive online analysis (SAMOA) [22]; 3) streaming graph analytics systems, such as GraphJet [23]; and 4) general purpose streaming data processing frameworks, such as Flink and Spark Streaming, with low-latency and a distributed parallel processing architecture. Apache Flink is an open-source distributed stream processing framework for stateful computations over unbounded and bounded data streams. Spark is a unified analytics engine for large-scale data processing supporting high-level APIs and general execution graphs.

Under several emerging application scenarios, such as operational monitoring of extensive infrastructure [24], IoT [25], and smart cities, data streams must be processed under very short delays, and the data volume is enormous [26]. These data stream processing frameworks have to be scalable and efficient. To meet these challenges, architecture has been proposed to use cloud computing to enable data stream processing as the resource elasticity and fault tolerance features of cloud computing. Here, describe several public cloud solutions for processing streaming data. Amazon Kinesis Streams [27] is a service that enables continuous data intake and processing for several types of applications, such as data analytics and reporting, infrastructure log processing, and complex event processing. Google Cloud Dataflow [28] is a programming model and managed service for developing and executing a variety of data processing patterns, such as extract, transform, and load (ETL) tasks, batch processing, and continuous computing. Azure stream analytics (ASA) enables real-time analysis of streaming data from several sources, such as devices, sensors, websites, social media, applications, and infrastructures, among other sources [29].

C. Amazon Cloud Services

Since serverless computing is featured as FaaS for cloud customers and BaaS for cloud vendors, we adopt the following three backend services in addition to AWS Lambda. We use Amazon S3 to store the raw data and AWS SQS as the communication channel between Lambdas and AWS DynamoDB to persist the results.

Amazon S3: Amazon Simple Storage Service (Amazon S3) is an object storage service allowing users to store, protect and retrieve data from “buckets” at any time from anywhere. Amazon S3 focuses on two key components: 1) buckets and 2) objects that work together to create the storage system. Users create buckets to store objects in the cloud. Objects are data

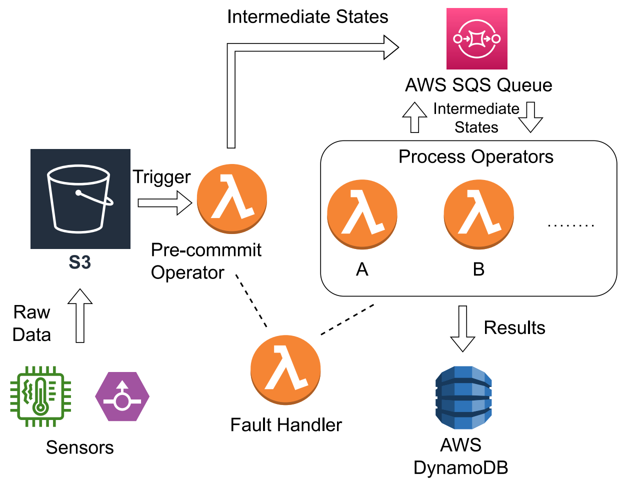


Fig. 2. Overview of the serverless stream computing framework.

files, including documents, photos, and videos, identified by a unique key. The use cases of Amazon S3 include data lakes, mobile applications, IoT devices, and big data analytics.

AWS SQS: AWS SQS is a managed service by AWS to handle message queueing, releasing developers from setting up and maintaining a queue system. AWS SQS is built on the broad mechanism of message queues and provides high-level APIs that developers can use to communicate with the service. SQS is frequently used to decouple distributed backend services or accommodate mismatches in service scalability.

AWS DynamoDB: DynamoDB is a NoSQL serverless database provided by AWS which follows a key-value store structure and adopts a distributed architecture for high availability and scalability. In DynamoDB, data is organized in tables containing items, and each item contains a set of key-value pairs of attributes. As in any serverless system, there's no infrastructure provisioning needed with DynamoDB.

III. Design

This section gives a detailed overview of SPSC, a stream processing framework atop serverless computing platforms.

A. Overview

The application mainly includes parallel workflow, communication, and fault tolerance. Fig. 2 shows the designed stream computing framework based on the AWS services, which reduces users' hardware requirements. In the hypothetical industrial scenario, the intelligent sensors collect and upload the raw data to the low-cost persistent storage service S3 provided by AWS. Generally speaking, production data analysis in industrial scenarios does not require too high-real time, and the raw data is persistent and massive. Using S3 can save considerable storage costs. When the raw data is uploaded to S3, the Lambda instance of the presubmit operator will be triggered to perform the initial processing of the data. Then the intermediate states of the data will be pushed into the queue of AWS SQS. The messaging mechanism of SQS itself can ensure that the intermediate state will only be processed by one Lambda instance, avoiding the additional cost caused by redundant processing. Similarly, the pushed message in the queue will trigger the start of the instance of the process operator. The process operator is the coding entry of the user program logic. The intermediate states between process operators are also transferred through SQS queues to ensure real time and nonrepetitive processing. The last process operator, which can be regarded as the exit of the user program, stores the results in the structured NoSQL database AWS DynamoDB, thus making the calculation results persistent. The framework should deal with and recover from the failure of computing nodes to deal with the unbounded input of stream computing. Hence, we design a Lambda function to handle failures in the computing procedure.

B. Parallel Workflow

First, the framework abstracts the computing task into the concept of the atomic stream. The data in the stream should be cut as small as possible to meet the concept of data elements. All kinds of operators in the framework operate based on the level of data elements, which are the lowest and indivisible processing levels.

Atomic Stream: Atomic stream is a relatively ordered, distributed, and immutable atomic stream. Processors generate atomic streams, and consumers consume them. To ensure the indivisibility of atoms, we stipulate that consumers must complete all the work of consuming and processing the atom it holds before consuming and processing the next atom. Consumption must follow atomic order. Producers must produce atoms after consuming and processing and before consuming and processing the subsequent ones. The resulting atomic production order will establish the atomic order, enabling us to combine the roles of producers and consumers safely. An operator can assume the roles of producers and consumers at the same time, reducing the difficulty of user coding while maintaining end-to-end fault tolerance between micro-services. In practical coding, atomic streams can be understood as data flowing between processors.

Workflow: Workflow consumes atomic stream and generates atomic stream. Workflow is a directed acyclic graph (DAG) of sources, procedures, and links. Workflow has a source, which is the event intake point of workflow. Procedures are stateful operators that use atoms and generate atoms. Workflow has a sink point, which can be considered as the exit of workflow: the exit of calculation results. The workflow needs to conform to the principles of atomic processing. It always uses and processes one atom at a time; it does not need to process two atoms simultaneously. Nested workflows are not supported. Workflow has an input source and an output sink, just like the source and sink in other stream computing systems. In short, a workflow can be understood as a defined function, requiring input (atomic stream) and returning output (atomic stream) after internal processing is completed.

Parallel Method: To improve the throughput and efficiency of stream computing, common parallel methods are task parallelism and data parallelism. Task parallelism allows tasks from different operators to perform calculations on the same or different data in parallel to better utilize the cluster's computing resources. Data parallelism performs the same operation in parallel on the data subset, allowing the processing of large amounts of data and spreading the computing load to multiple computing nodes. In the serverless architecture, the operators implemented by the Lambda function and triggered by the specified data flow naturally have the task parallel attribute. By cutting the processing process into multiple context-independent processing operators, the framework realizes the on-demand startup of operators, thus saving costs while ensuring throughput. The instantiation of lambda functions will automatically obtain instance resources from the public reservation pool to perform computing tasks and dynamically tilt computing resources for each function based on utilization. The feature of AWS SQS service that can only be seen by one person at the same time automatically manages the data subsets. The above features of AWS services enable the framework to automatically generate several operator instances to process atoms in parallel while avoiding some common problems of data parallelism. On the serverless architecture, users can code highly reliable stream computing logic by applying the idea of task parallelism and data parallelism without having too much relevant knowledge.

C. State Storage

The data processed by the stream processing system is often borderless: data will always be input from the data source, and users need to see the real-time results of SQL queries. At the same time, the computing nodes in the stream processing system may make errors and failures and expand and shrink in real time according to the user's needs. In this process, the system should efficiently transfer the intermediate states of the calculation between nodes and persist the results to the external system to ensure uninterrupted calculation. Common state storage solutions, such as embedded storage, require the computing nodes to manage the state storage, which is not applicable in the dynamically generated, stateless, and storage-and-calculation separation serverless architecture. The stateless nature of Lambda functions makes the serverless-based stream computing framework only adopt the architecture of storage and computing separation.

Separation of Storage and Computing: The storage responsibility and calculation responsibility of the system are separated. The storage node is only responsible for data storage, while the calculation node is only responsible for the calculation, that is, to execute business logic. Such a design is called the separation of storage and computing. Each instance is the same for the stateless computing instances generated by Lambda instantiation and naturally supports horizontal expansion. The generated instances of the same type obtain the states by polling and then process them to achieve load balancing easily. Failover is also simpler and faster. If an instance fails, the computing task on it will be acquired by other instances. Developers can focus on computing business logic without paying attention to such troublesome storage problems as data consistency, data reliability, and data read and write performance, significantly reducing development difficulty and improving development efficiency.

Message Queue: Stream computing systems usually do not need message queues because they can communicate directly between functions, and the end-to-end exactly once mechanism is also guaranteed in other ways. However, the lack of direct communication between instances on the serverless platform makes implementing the end-to-end exactly once mechanism challenging. The message queue service provided by the service provider, such as AWS SQS, can ensure that each message pushed to the queue will be shared by only one object at the same time. After a while, if the object does not perform other operations on the message, the message will be released to other objects. Set its visibility time to be longer than the instance lifetime in the framework; you can think that when a message is released to other objects, the previous instance has failed, thus realizing the failover. Therefore, in the framework's design, we use AWS SQS as the message queue to complete the communication between operators and intermediate state storage.

Persistent Storage: The final results of stream computing, or some states that do not need immediate processing temporarily, need to be persisted to the external storage system. For example, if we want to count the production data in the past five minutes, and some of the earlier data arrive later than the later data due to network communication and other reasons, in this case, we can only store the states in the persistent storage database, and then sort it before processing. The disordered and temporarily stored message queue obviously does not support the above requirements. NoSQL, which does not need to design data relations in advance, is easy to expand and can be used on demand, is widely used in persistent storage systems of stream computing systems. AWS DynamoDB, a fully hosted cloud NoSQL database service provided by Amazon, can realize seamless expansion and automatically delete expired items from the table. We use it as a framework for persistent storage system.

D. Fault Tolerance

The problem of unbounded input stream in the stream computing system has brought many new challenges to Fault Tolerance, such as low latency, exactly once mechanism, and so on. Many stream computing tasks are 7×24 h without interruption, with end-to-end second delay. It is extremely difficult to quickly recover to normal in case of unexpected problems, such as network flash, machine failure, and so on, without affecting the correctness of the calculation results. Moreover, the statelessness and separation of components of the serverless architecture make the fault-tolerant design of this framework different from the traditional stream computing system.

Timeout: Traditional stream computing system operators will exist for a long time once generated, while Lambda instances have a limited and short survival time. To save the cost of repeatedly generating instances, the framework is designed to continuously process atoms once an operator instance is generated until atoms cannot be obtained or times out. Therefore, it can be considered that the instance running timeout will be a common exception in the running process. Considering the startup costs that can be saved, we think it is acceptable that a very small amount of data is delayed due to timeout exceptions. We set a timeout exception handling function. When the instance senses that it is about to timeout, it will throw a timeout exception and invoke the function. The exception handler invokes the operator function again according to the received event. In fact, because of the high substitutability of operator instances, even if the function is not restarted, other working instances will poll the processing atoms. However, restarting it makes the number of concurrent instances of the operator stable and can reduce the fluctuation of throughput as much as possible.

At Least Once: Due to the statelessness and nondirect communication of Lambda instance running, the snapshot recovery mechanism commonly used in stream computing systems, such as Flink, is obviously difficult to implement. That is to say, it is very difficult to achieve Exactly Once on the serverless architecture which is hard to perceive and maintain the running state. However, the design of storing intermediate states through message queues and serving as communication channels between operators makes it easy for the framework to implement At Least Once. AWS SQS guarantees that a message will only be owned by one object at the same time. When the visibility time setting is exceeded, if the message has not been processed by the owner, it will be returned to the queue and opened to other objects. By setting the visibility time slightly longer than the instance survival time, we can ensure that each atom will be processed at least once.

IV. Evaluation

In order to verify the performance of the prototype of our framework, we compare its performance with that of the serverless real-time computing platform launched by Alibaba Cloud in this section.

A. Environment

We deploy the framework prototype on the AWS platform and use the services provided by the supplier. As designed in the framework, we create AWS SQS queues and AWS DynamoDB tables for the prototype. Then, we create S3 buckets to store randomly generated source data. Next, we deploy the prototype code on AWS Lambda and allocated a concurrent quota of 40 instances. Among them, to improve the throughput of reading data from slow S3 storage, ten reserved concurrent accesses is allocated for the precommit operator of the source. In other words, there are still 30 instances left in the public reservation pool.

For the Flink version of Alibaba Cloud's real-time computing platform, we have rented it in a pay-as-you-go way. According to the official manual, 1 CU = 1 core CPU + 4 GB memory. CU corresponds to the CPU computing capacity of the underlying system. When creating a workspace, the system deploys a development console for each cluster. Each development console and its necessary components require about 2 CU of control resources. So we rented 6 CU, 2 for controlling resources and 4 for the actual calculation. Other configurations, such as the database RDS, use the default configuration 4 RCU.

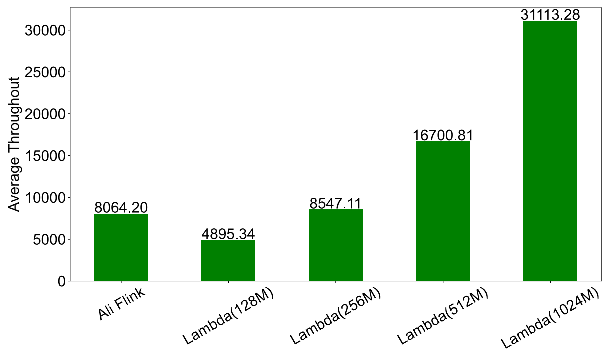
B. Performance Metrics

We have carried out the same experiments on the prototype and the leased Alibaba Cloud real-time computing platform, and the evaluation metrics of their performance are as follows.

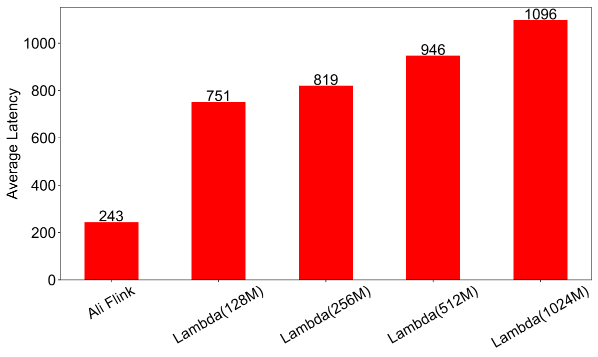
Throughout: The number of events successfully transmitted by the computing framework in unit time. The unit of throughput in this experiment is events/second. Throughput reflects the system's load capacity and how much data the system can process per unit of time under the corresponding resource conditions. Throughput is often used for resource planning but also to help analyze system performance bottlenecks to make corresponding resource adjustments to ensure that the system can meet the processing capacity required by users. Suppose the merchant can make 20 lunches per hour (throughput: 20 lunches per hour), and a delivery boy can only deliver two lunches per hour (throughput: 2 lunches per hour). The bottleneck of this system is in the delivery of the boys, which can arrange ten delivery boys for the merchant.

Latency: The time of the event from entering the system to exiting the system. The unit of latency in this experiment is a microsecond. Latency reflects the real-time processing of the system. Many real-time computing services, such as financial transaction analysis, have high requirements for latency. The lower the latency, the stronger the real-time data. Suppose it takes 5 min for the merchant to make lunch and 25 min for the brother to deliver. In this process, the user feels a latency of 30 min. If the latency becomes 60 min after changing the delivery plan, and the food is cold when delivered, the new plan is unacceptable.

Cost: General ledger of all cloud services leased. Since the cloud service leased in this experiment comes from two cloud service providers in different regions, and the currency charged is their local official currency, we have converted the two according to the exchange rate at the time of the experiment, which is about 6.7 yuan to 1 USD. The charging standard of Alibaba Cloud's real-time computing platform is 0.133 USD/CU/hour, and the bill is calculated from the time the workspace is generated. It also includes the SLB service and database service it provides. For SLB, the unit price of each instance is 0.01 USD per hour. RDS is used by default for databases, and its price is 0.055 USD/hour/RCU. Furthermore, the price per million write request units is 1.20 USD. The billing of AWS cloud services includes the billing of Lambda, SQS, and DynamoDB. For Lambda, when the memory is 128M, the price for each instance to run 1 ms is 2.1×10^{-9} USD. When the memory is expanded proportionally, the price can also increase proportionally. For SQS, the first 1 million requests are free, and the price of 1 million to 100 billion requests is 0.40 USD per million requests. The number of demand requests for our framework should be within the latter range. For DynamoDB, the price per million write request units is 1.25 USD.



(a)



(b)

Fig. 3. Average results obtained by counting several random windows. (a) Throughput of Alibaba Flink and prototype. (b) Latency of Alibaba Flink and prototype.

TABLE I
COST CALCULATION

Scenario	Cost of Running for 1 hour(USD)	Cost without Writing to DB(USD)	Cost per million Events(USD)
Ali Flink	36.392	1.556	0.053
Lambda (128M)	20.433	0.6516	0.053
Lambda (256M)	42.894	1.476	0.041
Lambda (512M)	76.653	2.669	0.0445
Lambda (1024M)	118.153	4.249	0.0466
Lambda (2048M)	178.015	6.691	0.0488

C. Performance

In order to test the performance of the framework itself, we conducted input-output tests on both. Specifically, in the prototype, we set two operators: 1) the precommit operator and 2) the processing and sink operator. The precommit operator randomly generates a message, records the timestamp, and then pushes it into the SQS queue. The processing operator writes the processing time after receiving the message from the queue, which is the output time, and then writes it to the table. The same applies to Alibaba Cloud's real-time computing platform, where a job that generates and writes data to the database is published. Then, we randomly grab several 5-min windows and count the average throughput and latency of the two during this period.

Fig. 3 shows the throughput and latency of the experimental results. As seen from Fig. 3(a), the throughput of the prototype we built has no significant defects compared with Ali Flink. It can achieve a considerable increase in throughput with the increase of allocated RAM. Throughput does not increase linearly. It is speculated that communication with SQS limits its linear growth. Therefore, the relationship between communication queues and throughput can be studied. Compared with Ali Flink, the latency of the prototype has increased significantly, which is understandable. The time cost of communicating with external cloud services, such as SQS, must be higher than the internal communication of main memory in Ali Flink. Comparing performance between different configurations is not significant, but it proves that our framework can realize the functions of a stream computing system.

In order to compare the advantages of our framework, we calculated the costs that customers usually care about most, as shown in Table I. Much overhead comes from reading and writing data to the database because persistent storage is always expensive. However, generally speaking, the request to write records to the database is far lower than the request in the input-output experiment, so the cost calculation of database storage should be removed. From the perspective of processing costs per million businesses, our framework has a slight advantage over Ali Flink; that is, it can save 10.8% of the cost on average when processing the same business volume. Although throughput and cost are not strictly linear, we can still estimate the approximate throughput of the prototype at the same cost by linear interpolation. We estimate that if Lambda's overhead cost is 1.028 USD, its estimated throughput is 8812 events/s. Our framework is expected to improve the performance by 10.12% at the same cost.

V. RELATED WORK

A. Serverless Computing and Its Application

Serverless computing [7], [8], [9] is a promising paradigm of next-generation cloud computing. It has been widely used in many fields thanks to its more lightweight virtualization runtime and faster startup and destruction time than virtual machines. In distributed machine learning and deep neural network training [30], the parallel ability that serverless computing can provide can optimize the training speed of data parallelism or pipeline parallelism [31], [32], [33], [34], [35]. LambdaML Jiang et al. [33] is a general machine learning training platform atop serverless computing, and conduct a comprehensive study on the different aspects, like communication channel, synchronization, and cost efficiency. Siren [35] also employs AWS Lambda, the most representative serverless computing platform, for distributed model training and designs a reinforcement learning algorithm to guide the resource configuration for functions. In addition, serverless computing can also be applied to deploying the pretraining model [36], [37], [38], [39] to deal with the inference task of changing requests. Because serverless computing has good scalability and an efficient startup rate, the operation and maintenance personnel do not need to consider the system's load balancing and concurrent processing when deploying model reasoning tasks. In addition to the training and deployment tasks of machine learning, other high-performance computing tasks can also benefit from serverless computing platforms, such as video processing [40], [41], [42], [43], high-dimensional matrix operations [44], [45], workflow tasks [46], [47], [48], etc.

B. Serverless Computing for Data Processing

Reference [49] introduces a serverless architecture for big data analysis. As the data size is increasing daily, it is tough and complex to design the exact architecture for data analysis, including server management, storage, clustering, algorithm deployment, etc. The misconfiguration would lead to the underuse of resources and infrastructure and unnecessarily high costs. With the challenges of scalability and efficiency that big data processing systems face, researchers have turned to serverless (Fuction-as-a-Service) to strengthen big data analysis. Nastic et al. [6] proposed a serverless real-time data analytics platform for edge computing. As the user-defined functions are seamlessly and transparently hosted and managed by the serverless platform, it releases the developers of the burden of resorting to optimal management of underlying infrastructure on the edge side. Rahman and Hasan [49] presented the serverless architecture for big data analytics with a serverless extensive data application on AWS. With the serverless paradigm, developers can concentrate on implementing data analytics applications rather than underlying infrastructure and pay only for consistent execution rather than particular server components. Portals [50] is a serverless, distributed programming model that blends the exactly once processing guarantees of stateful dataflow streaming frameworks with the message-driven compositionality of actor frameworks. With Portals, the decentralized application can be built dynamically and scale on demand, guaranteeing strict atomic processing.

VI. Conclusion

In this article, we proposed SPSC, a stream computing framework built on serverless architecture to cope with industrial data. By dividing and abstracting the events into atoms and atomic streams, SPSC realizes task and data parallelism and achieves high throughput and efficiency of stream computing. Combined with AWS components, such as AWS SQS and AWS DynamoDB, SPSC achieves abilities of at least once guarantee and persistent storage for stream computing applications in the serverless computing environment. Through extensive evaluation, we show that SPSC exceeds Alibaba's real-time computing Flink version by 10.12% in performance.

References

[1] W. Li, Y. Liang, and S. Wang, *Data Driven Smart Manufacturing Technologies and Applications*. Cham, Switzerland: Springer, 2021.

[2] A. Kusiak, "Smart manufacturing," *Int. J. Prod. Res.*, vol. 56, nos. 1–2, pp. 508–517, 2018.

[3] C. Zhang et al., "When autonomous systems meet accuracy and transferability through AI: A survey," *Patterns*, vol. 1, no. 4, 2020, Art. no. 100050.

[4] R. Ren, T. Hung, and K. C. Tan, "A generic deep-learning-based approach for automated surface inspection," *IEEE Trans. Cybern.*, vol. 48, no. 3, pp. 929–940, Mar. 2018.

[5] T.-M. Choi, H. K. Chan, and X. Yue, "Recent development in big data analytics for business operations and risk management," *IEEE Trans. Cybern.*, vol. 47, no. 1, pp. 81–92, Jan. 2017.

[6] S. Nastic et al., "A serverless real-time data analytics platform for edge computing," *IEEE Internet Comput.*, vol. 21, no. 4, pp. 64–71, Jul. 2017.

[7] Z. Li, L. Guo, J. Cheng, Q. Chen, B. He, and M. Guo, "The serverless computing survey: A technical primer for design architecture," *ACM Comput. Surv.*, vol. 54, no. 10s, pp. 1–34, 2022.

[8] H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless computing: A survey of opportunities, challenges, and applications," *ACM Comput. Surv.*, vol. 54, no. 11s, pp. 1–32, 2022.

[9] E. Jonas et al., "Cloud programming simplified: A berkeley view on serverless computing," 2019, *arXiv:1902.03383*.

[10] S. S. Manvi and G. K. Shyam, "Resource management for infrastructure as a service (IaaS) in cloud computing: A survey," *J. Netw. Comput. Appl.*, vol. 41, pp. 424–440, May 2014.

[11] R. Yasrab, "Platform-as-a-service (PaaS): The next hype of cloud computing," 2018, *arXiv:1804.10811*.

[12] A. Agache et al., "Firecracker: Lightweight virtualization for serverless applications," in *Proc. 17th USENIX NSDI*, 2020, pp. 419–434.

[13] "OpenWhisk." Accessed: Jan. 2023. [Online]. Available: https://openwhisk.apache.org/

[14] "OpenFaaS." Accessed: Jan. 2023. [Online]. Available: https://www.openfaas.com/

[15] "Fission." Accessed: Jan. 2023. [Online]. Available: https://fission.io/

[16] T. Akidau et al., "The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1792–1803, 2015.

[17] H. Shi et al., "Robust searching-based gradient collaborative management in intelligent transportation system," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 20, no. 2, pp. 1–23, 2023.

[18] H. Guo et al., "Siren: Byzantine-robust federated learning via proactive alarming," in *Proc. ACM Symp. Cloud Comput.*, 2021, pp. 47–60.

[19] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang, "NiagaraCQ: A scalable continuous query system for Internet databases," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2000, pp. 379–390.

[20] S. Chandrasekaran et al., "TelegraphCQ: Continuous dataflow processing," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2003, p. 668.

[21] S. Alsubaiee et al., "AsterixDB: A scalable, open source BDMS," 2014, *arXiv:1407.0454*.

[22] G. D. F. Morales and A. Bifet, "SAMOA: Scalable advanced massive online analysis," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 149–153, 2015.

[23] A. Sharma, J. Jiang, P. Bommannavar, B. Larson, and J. Lin, "GraphJet: Real-time content recommendations at twitter," *Proc. VLDB Endow.*, vol. 9, no. 13, pp. 1281–1292, 2016.

[24] Y. Tang, X. Xing, H. R. Karimi, L. Kocarev, and J. Kurths, "Tracking control of networked multi-agent systems under new characterizations of impulses and its applications in robotic systems," *IEEE Trans. Ind. Electron.*, vol. 63, no. 2, pp. 1299–1307, Feb. 2016.

[25] R. Zhang et al., "OSTTD: Offloading of splittable tasks with topological dependence in multi-tier computing networks," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 2, pp. 555–568, Feb. 2023.

[26] Z. Wang, R. Ma, H. Shi, L. Lin, and H. Guan, "Batch gradient descent-based optimization of WMMSE for rate splitting strategy," in *Proc. Int. Symp. Wireless Commun. Syst. (ISWCS)*, 2022, pp. 1–6.

[27] (Amazon, Seattle, WA, USA). *Amazon Kinesis Streams*. Accessed: Jan. 2023. [Online]. Available: https://aws.amazon.com/cn/kinesis/data-streams/

[28] (Google, Mountain View, CA, USA). *Google Cloud Dataflow*. Accessed: Jan. 2023. [Online]. Available: https://cloud.google.com/dataflow/

[29] (Microsoft Corp., Redmond, WA, USA). *Azure Stream Analytics*. Accessed: Jan. 2023. [Online]. Available: https://azure.microsoft.com/en-us/services/stream-analytics/

[30] Q. Jiang et al., "Neural network aided approximation and parameter inference of non-Markovian models of gene expression," *Nat. Commun.*, vol. 12, no. 1, p. 2618, 2021.

[31] Y. Liu et al., "FuncPipe: A pipelined serverless framework for fast and cost-efficient training of deep learning models," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 3, pp. 1–30, 2022.

[32] P. G. Sarroca and M. Sánchez-Artigas, "MLLess: Achieving cost efficiency in serverless machine learning training," 2022, *arXiv:2206.05786*.

[33] J. Jiang et al., "Towards demystifying serverless machine learning training," in *Proc. Int. Conf. Manag. Data*, 2021, pp. 857–871.

[34] F. Xu, Y. Qin, L. Chen, Z. Zhou, and F. Liu, "λDNN: Achieving predictable distributed DNN training with serverless architectures," *IEEE Trans. Comput.*, vol. 71, no. 2, pp. 450–463, Feb. 2022.

[35] H. Wang, D. Niu, and B. Li, "Distributed machine learning with a serverless architecture," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2019, pp. 1288–1296.

[36] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "Optimizing inference serving on serverless platforms," *Proc. VLDB Endow.*, vol. 15, no. 10, pp. 2071–2084, 2022.

[37] K. Mahajan and R. Desai, "Serving distributed inference deep learning models in serverless computing," in *Proc. IEEE 15th Int. Conf. Cloud Comput. (CLOUD)*, 2022, pp. 109–111.

[38] J. Li, L. Zhao, Y. Yang, K. Zhan, and K. Li, "TETRIS: Memory-efficient serverless inference through tensor sharing," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2022, pp. 473–488.

[39] Y. Yang et al., "INFless: A native serverless system for low-latency, high-throughput inference," in *Proc. 27th ACM Int. Conf. Archit. Support Program. Languages Oper. Syst.*, 2022, pp. 768–781.

[40] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter, "Sprocket: A serverless video processing framework," in *Proc. ACM Symp. Cloud Comput.*, 2018, pp. 263–274.

[41] M. Zhang, Y. Zhu, C. Zhang, and J. Liu, "Video processing with serverless computing: A measurement study," in *Proc. 29th ACM Workshop Netw. Oper. Syst. Support Digit. Audio Video*, 2019, pp. 61–66.

[42] F. Romero, M. Zhao, N. J. Yadwadkar, and C. Kozyrakis, "Llama: A heterogeneous & serverless framework for auto-tuning video analytics pipelines," in *Proc. ACM Symp. Cloud Comput.*, 2021, pp. 1–17.

[43] M. Zhang, F. Wang, Y. Zhu, J. Liu, and B. Li, "Serverless empowered video analytics for ubiquitous networked cameras," *IEEE Netw.*, vol. 35, no. 6, pp. 186–193, Nov./Dec. 2021.

[44] V. Shankar et al., "Serverless linear algebra," in *Proc. 11th ACM Symp. Cloud Comput.*, 2020, pp. 281–295.

[45] V. Gupta, S. Kadhe, T. Courtade, M. W. Mahoney, and K. Ramchandran, "OverSketched Newton: Fast convex optimization for serverless systems," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, 2020, pp. 288–297.

[46] A. Mahgoub et al., "WISEFUSE: Workload characterization and DAG transformation for serverless workflows," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 2, pp. 1–28, 2022.

[47] A. Mahgoub, K. Shankar, S. Mitra, A. Klimovic, S. Chaterji, and S. Bagchi, "SONIC: Application-aware data passing for chained serverless applications," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2021, pp. 285–301.

[48] A. Mahgoub, E. B. Yi, K. Shankar, S. Elnikety, S. Chaterji, and S. Bagchi, "ORION and the three rights: Sizing, bundling, and prewarming for serverless DAGs," in *Proc. 16th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2022, pp. 303–320.

[49] M. M. Rahman and M. H. Hasan, "Serverless architecture for big data analytics," in *Proc. Glob. Conf. Adv. Technol. (GCAT)*, 2019, pp. 1–5.

[50] J. Spenger, P. Carbone, and P. Haller, "Portals: An extension of dataflow streaming for stateful serverless," in *Proc. ACM SIGPLAN Int. Symp. New Ideas, New Paradigms, Reflect. Program. Softw.*, 2022, pp. 153–171.

Zinuo Cai received the Doctoral degree in software engineering from Shanghai Jiao Tong University, Shanghai, China, in 2021, where he is currently pursuing the graduate degree in computer science.

His research interests are focused on resource schedule and system security in cloud computing.

Zebin Chen is currently pursuing the Master's degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China.

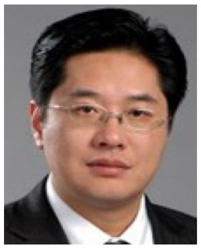
His research interests center around the applications of distributed systems.

Xinglei Chen is currently pursuing the Master's degree in computer science with Shanghai Jiao Tong University, Shanghai, China.

His research interests are focused on system security in cloud computing.

Ruhui Ma (Member, IEEE) received the Ph.D. degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2011.

He is currently an Associate Professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include cloud computing systems, AI systems, and machine learning.



Haibing Guan (Member, IEEE) received the Ph.D. degree from Tongji University, Shanghai, China, in 1999.

He is currently a Professor with the School of Electronic Information and Electronic Engineering, Shanghai Jiao Tong University, Shanghai, and the Director of the Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University. His research interests include cloud/distributed computing and machine learning.

Rajkumar Buyya (Fellow, IEEE) received the Ph.D. degree in computer science and software engineering from Monash University, Melbourne, VIC, Australia in 2002. Dr. Buyya is a Redmond Barry Distinguished Professor and the Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, The University of Melbourne, Melbourne, VIC, Australia. He has authored more than 625 publications and seven text books.

Dr. Buyya is one of the Highly Cited Authors in Computer Science and Software Engineering Worldwide (H-index = 153, G-index = 324, and more than 121 200 citations). Microsoft Academic Search Index ranked him as #1 Author in the World (2005–2016) for both field rating and citations evaluations in the area of distributed and parallel computing. He is recognized as a "Web of Science Highly Cited Researcher" from 2016 to 2021 by Thomson Reuters.