Elastic Stream Processing for Distributed Environments

Christoph Hochreiner, Stefan Schulte, and Schahram Dustdar • *TU Wien, Austria*

Freddy Lecue • *IBM Research–Ireland*

The Internet of Things introduces the need for more flexibility in stream processing. To address these challenges, the authors propose elastic stream processing for distributed environments. This novel concept allows for scalable and more flexible solutions compared to traditional approaches.

Big Data has become an immanently important topic in many application areas such as smart cities, smart factories, the smart grid, or smart mobility. Gartner estimates that there are already 1.1 billion connected physical devices alone for smart cities in 2015, and expects this number to rise to almost 10 billion devices in 2020 (www.gartner.com/newsroom/id/3008917). These connected devices form the so-called *Internet of Things* (IoT).

Although the IoT enables new opportunities in different industries and application areas, its potential benefits are only realizable once important research questions in the field of Big Data have been answered. These questions range from data access, storage, discovery, analysis, or processing to reasoning on Big Data. Especially *smart systems* — such as smart cities and the IoT — require real-time processing capabilities (including stream processing) to provide value-added services.[1] Although stream processing has been an active research area for several years, the topic is gaining momentum as the emerging IoT increases the amount of continuous streaming data (such as sensor data) to previously unknown levels.

IoT-based value-added services are triggered by sensor events or user requests, which lead to variable system loads being handled by stream processing systems. To cope with these challenges, a stream processing system must be elastic in terms of its processing capabilities.

Regarding elasticity, we consider three interdependent dimensions. The first dimension is *quality elasticity*. This deals with aspects such as the response time, the quality of a response for queries against streaming data, or the amount of successfully processed streaming data. If too much data must be handled by a stream processing system, this can lead to a reduction in service quality[2] or to all non-processable data being dropped.

The second dimension is *resource elasticity*. It indicates that the amount of computational resources required to process streaming data must be adapted at runtime to cope with the current system load.[3] Until now, most stream processing systems ran on fixed computational resources in one location, such as a data center, and could distribute the workload only among these fixed resources. In fact, systems with fixed resources could adapt to volatile data rates only by reducing the quality of service (QoS).

The third dimension is *data elasticity*. This means that data aren't necessarily stored or processed locally, but rather in various remote data repositories in the Cloud. NoSQL approaches, such as MongoDB (www.mongodb.org), already envision the distributed storage of huge amounts of data, but until now, stream processing hasn't applied these NoSQL concepts.

To take stream processing to the next level, resource elasticity is a prerequisite to maintain service-level agreements (SLAs), regardless of the system load. It's also an efficient approach to elastically deal with huge amounts of stored data located in different geographical locations. In order to back up our claims, we provide a scenario from the mobility domain (see the related sidebar), propose elastic stream processing for distributed environments, and identify several open research challenges. Let's begin by taking a closer look at the proposed model.

Conceptual Overview

To cope with the changing resource requirements and different geographical locations of sensors as well as the stream processing nodes, we propose an elastic stream processing model for distributed environments. Our model is composed of multiple self-contained nodes (see Figure 1).

As we discuss in the sidebar "Related Work in Stream Processing," in the past stream processing systems generally pursued a monolithic approach where a centralized component manages the communication and execution of stream processing operations. These operations range from filtering data, to querying sensor data for a specific time span, to customizing business logic (for example, transforming sensor data into other data formats).

Monolithic systems are deployed in one geographic location (on a cluster, for example), which eases the communication among the different processing nodes as well as the resource management for the processing operations. In the past, static stream processing scenarios with short and reliable communication paths successfully applied this centralized approach. Nevertheless, this approach is unsuitable for the IoT, because IoT scenarios are likely to evolve over time, in the sense that a smart city is constantly evolving. This leads to continuous changes of the IoT landscape, and therefore also to the stream processing infrastructure. To counter the IoT domain's challenges, we must design an elastic stream processing model consisting of self-contained nodes that we can orchestrate during runtime.

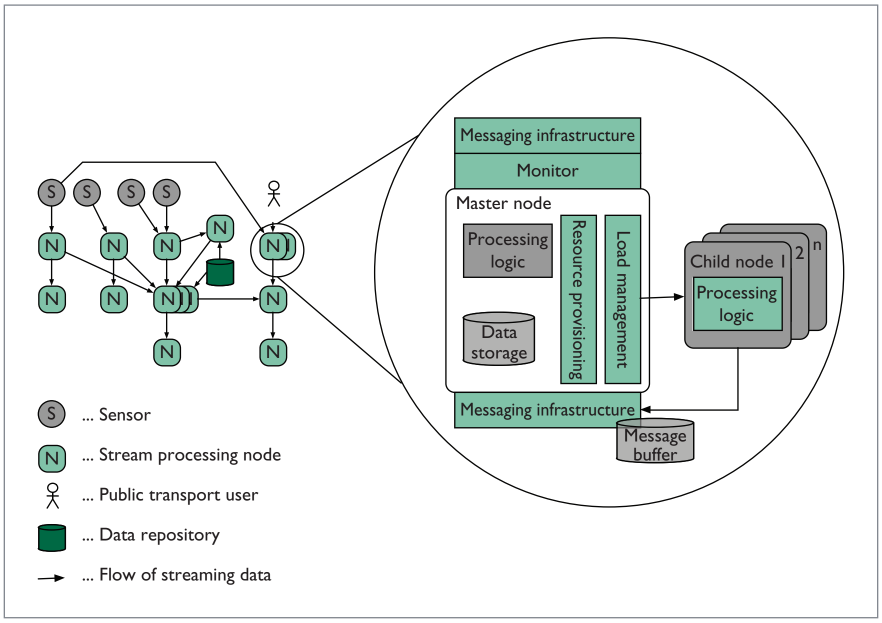

Figure 1. Elastic stream processing model. On the left we show the choreography based on a motivational scenario, and on the right we show a processing node in detail.

The left side of Figure 1 shows an exemplary choreography based on a motivational scenario, and the right side presents one of the processing nodes in detail. Such a node is deployed on fixed computational resources (for instance, in a cloud-based infrastructure), where each node can autonomously allocate computational resources according to the data rates. To realize elastic scaling, the node can allocate additional resources — obtaining, for example, virtual machines (VMs) as child nodes — and then it can deploy processing operations on these VMs.

Besides the elastic data processing capabilities, each single node also maintains data storage (such as a database) to realize stateful operations (such as data aggregations) as well as a messaging infrastructure to receive data from sensors or preceding nodes. For the outgoing communication, the node further maintains a message buffer to ensure that no information is lost, if communication to succeeding nodes isn't possible due to temporary technical failures or environmental aspects, such as communication dead spots. The node's last component is the monitor, which records changing data rates over time to allow predictive reasoning for the identification of future computational resource requirements.

Open Research Challenges

Even though elastic stream processing builds on established principles and concepts, four important research challenges (detailed in the following) must be addressed to achieve widespread adoption. These topics include methodologies and algorithms as well as the software tools necessary to realize elastic stream processing for distributed environments.

Cost-Efficient Resource Provisioning

Although there has been much research toward the optimal deployment of stream processing operations on fixed resources, there are hardly any advances in leveraging the resource elasticity provided by cloud-based resources. Because the system load varies over time due to varying data rates, a stream processing system may run into over- or underprovisioning

Mobility Domain Scenario

Our motivation for elastic stream processing originates from the omnipresent trend towards smart cities. Smart cities rely on numerous sensors that cover different aspects of the city's management and interaction with citizens, such as smart energy, smart homes, and smart transport networks.[1] These smart systems form a distributed environment. To demonstrate the challenges for stream processing, we consider an intelligent transportation system (ITS).[2] The ITS is responsible for monitoring and directing public transport as well as managing related infrastructure, such as information displays. The ITS depends on different data sources — for example, GPS sensors mounted on public transport vehicles, traffic information (such as induction loops measuring the number of cars driving by), or weather information.

Figure A illustrates the orchestration of the ITS's different data sources and stream processing operations. Four value-adding services are depicted that are relevant for system users. The first two services represent simple services that notify the driver of a public transport vehicle about the latest traffic information. Each service consists of two stateless processing operations (see Figure A). The first operation filters traffic sensor data and the second operation propagates the information to the drivers.

More complex services, such as the update service for displays, obtain information from all four data sources, an additional operation (which calculates public transport vehicles' movement speed based on GPS locations over time), as well as routing information. The routing information is obtained from a cloud-based data repository. Movement speed and sensor information are then forwarded to the prediction operation, which reasons on the data. This prediction operation is stateful, because it stores historic predictions over a longer time span to provide better prediction results.

Besides the operational components, the ITS also provides real-time routing information for public transport. Public transport users initiate routing queries and three stateless operations process each query to obtain the desired result for the user.

The individual sensors' data rates (the amount of information provided by the sensor at a specific point in time) changes throughout the course of the day, mainly because of the commuters' different driving directions. These volatile data rates challenge a stream processing system to cope with varying resource requirements to process streaming data.

The stream processing system further obtains the information from different sensors and data repositories (such as roadmaps), which are located in different geographical locations. Because the data originate from different infrastructures, the data might be inconsistent across various sources: A GPS device reports a bus driving on a specific street while an induction loop on that same street detects a traffic jam.

When processing streaming data from various sources, it's challenging to select the most accurate and reliable data source. In terms of cost and performance optimization, it's desirable to process the data geographically from the closest location. This reduces the effort required, such as data transfer costs that are required to transfer streaming data.

This scenario provides a brief glance at services and challenges from the mobility domain. Similar use cases also arise in other domains, such as building automation for smart cities, which involves monitoring countless sensors and processing their data to ensure a safe and pleasant environment. We further envision similar challenges in other areas such as e-healthcare, where systems must process large amounts of data in real time to gain insight into a patient's time-dependent information, such as heart rate or temperature.

References

1. L. Fei et al., "Web-Scale Service Delivery for Smart Cities," *IEEE Internet Computing*, vol. 17, no. 4, 2013, pp. 78–83.
2. A. Biem et al., "Real-Time Traffic Information Management Using Stream Computing," *IEEE Data Eng. Bullet*, vol. 33, no. 2, 2010, pp. 64–68.

scenarios, if the system only has a fixed amount of resources at its disposal. For instance, in off-peak times, a stream processing system could run into an overprovisioning scenario, because not all available computational resources are used. However, a stream processing system may also run into an underprovisioning scenario at peak times, when the stream-processing system can't process all incoming requests or streaming data, and therefore violates SLAs. The cloud computing paradigm[4] provides a promising solution by replacing fixed resources with elastic ones. A resource-elastic stream processing system leases and releases the required computational resources on demand to minimize the operational costs while guaranteeing a high QoS.

Although the introduction of elastic resources provides a solution to resource-related challenges, the ad hoc nature of such resource-allocation approaches introduces a dimension of complexity for deploying stream processing operations. Also, the leasing and releasing strategy must be optimized for real-world requirements, such as the billing time unit (the minimal leasing duration for cloud resources) and historical data rates. Such data rates allow the stream processing system to predict future resource requirements and optimize allocation of elastic resources just in time to avoid underprovisioning scenarios.

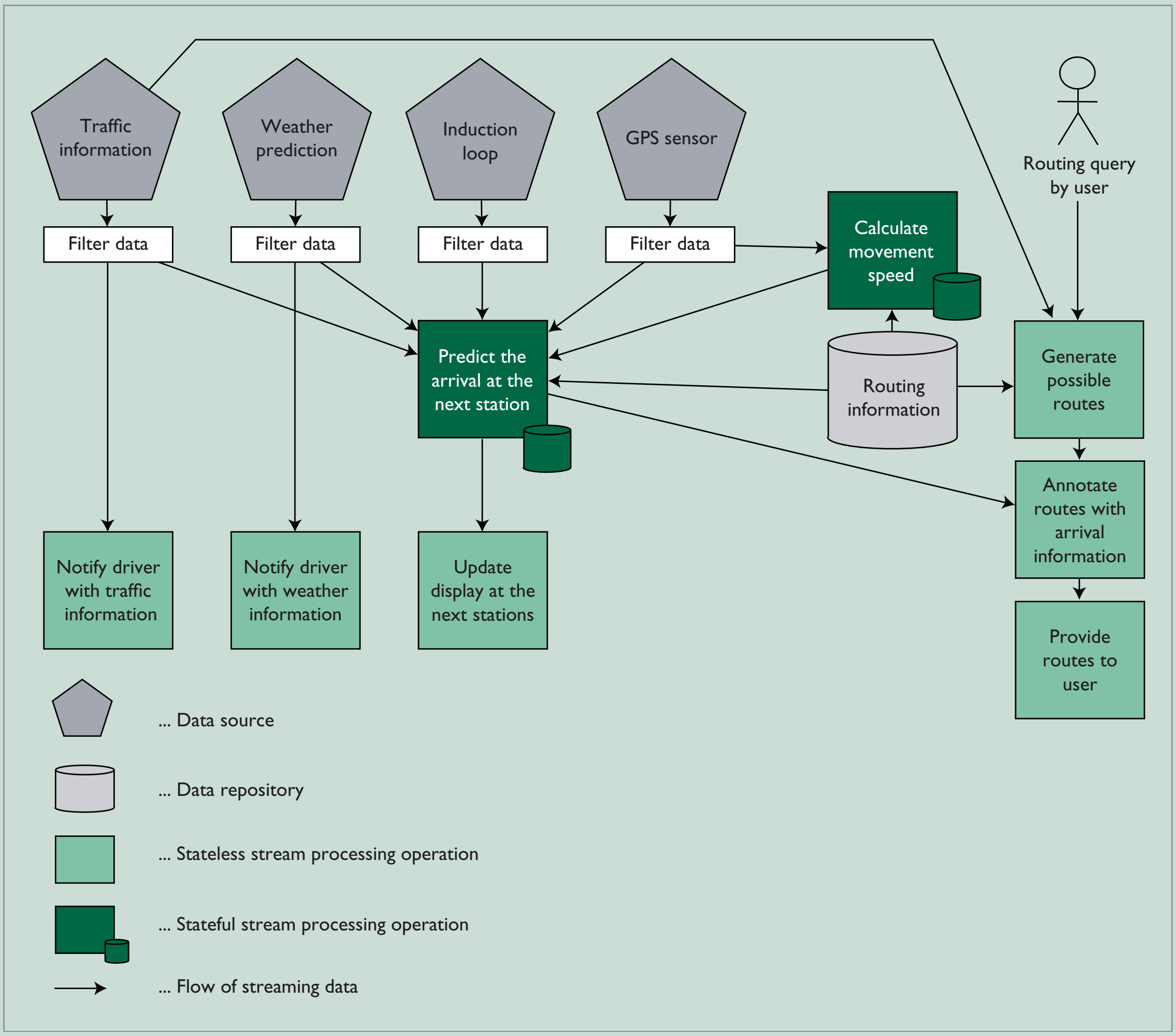
Traffic information
Weather prediction
Induction loop
GPS sensor
Routing query by user
Filter data
Filter data
Filter data
Filter data
Calculate movement speed
Predict the arrival at the next station
Routing information
Generate possible routes
Annotate routes with arrival information
Provide routes to user
Notify driver with traffic information
Notify driver with weather information
Update display at the next stations
... Data source
... Data repository
... Stateless stream processing operation
... Stateful stream processing operation
... Flow of streaming data

Figure A. Motivational scenario. Orchestrating the different data sources and stream processing operations provided by an intelligent transportation system (ITS).

SLAs' Data Stream Processing

Because quality elasticity represents one of the elasticity dimensions for stream processing, we must design appropriate SLAs to assess the QoS for elastic stream processing. Until now, stream processing only considered the *latency* of processed streaming data or the *ratio of dropped streaming data* in comparison to all streaming data.

Besides such basic SLAs, there are other areas that SLAs can cover: First, SLAs can be applied to queries on streaming data, making it possible to relax the result's quality in a controlled manner by shortening the observation time span or omitting relevant query data.

Second, SLAs can be assigned to individual streaming data, such as privacy restrictions or stating a validity period. In some domains, the information provided by sensors is transient and the information may be outdated after a few seconds. Therefore, it's possible to implement a context-sensitive load-shedding mechanism that discards all outdated data, instead of randomly discarding data to reduce the load.

Efficiently Placing and Migrating Stream Processing Nodes

Optimal placement of stream processing nodes is crucial for performance- and cost-efficient stream processing in the IoT. Different deployments are possible for stream processing nodes — for instance, on an embedded infrastructure with limited processing and

Related Work in Stream Processing

Because data rates change over time—in some cases even drastically—processing this data in real time isn't a straightforward task. To process streaming data, different stream processing systems such as Aurora,[1] Borealis,[2] MillWheel,[3] or IBM System S[4] provide solutions.

Early stream processing systems like Borealis[2] originate from the database-management domain. From a high-level perspective, such systems extend the relational database model to support the continuous aspects of streaming data. The Borealis architecture considers different processing nodes—such as software components that process streaming information—to enable parallel data processing for large amounts of data. However, the architecture only considers quality elasticity in terms of load shedding to deal with load peaks.

More advanced stream processing systems such as IBM System S[4] reliably and efficiently process different kinds of streaming data (such as financial or telecommunication data). Although the processing nodes support a distributed deployment (on a cluster, for example), IBM System S still maintains several centralized components, which manage monitoring, communication, and failure recovery. Besides research prototypes like Borealis, and proprietary stream processing systems such as IBM System S, there are also community-driven frameworks like Apache Storm (https://storm.apache.org) or Apache S4 (http://incubator.apache.org/s4/). Both frameworks can process huge amounts of data for Web-based services, to process social media data, for example. The architecture of these frameworks is based on a central component that manages data flow. Even though these stream processing frameworks feature a cluster-based deployment, they don't support resource elasticity for single processing nodes or an orchestration of processing nodes during runtime.

More recently, Benjamin Satzger and his colleagues[5] proposed an elastic stream computing platform for the Cloud, which provides an easy-to-use programming model to leverage the resource elasticity of the Cloud to deal with volatile data rates. However, their prototype doesn't address the challenges of accessing relevant data from different cloud infrastructures. Instead, they focus on elastically allocating computational resources in the Cloud for one static data stream processing application.

References

1. D. Carney et al., "Monitoring Streams: A New Class of Data Management Applications," *Proc. 28th Int'l Conf. Very Large Data Bases*, 2002, pp. 215–226.
2. D.J. Abadi et al., "The Design of the Borealis Stream Processing Engine," *Proc. Conf. Innovative Data Systems Research*, vol. 5, 2005, pp. 277–289.
3. T. Akidau et al., "MillWheel: Fault-Tolerant Stream Processing at Internet Scale," *Proc. Very Large Data Bases Endowment*, vol. 6, no. 11, 2013, pp. 1033–1044.
4. L. Amini et al., "SPC: A Distributed, Scalable Platform for Data Mining," *Proc. 4th Int'l Workshop on Data Mining Standards, Services, and Platforms*, 2006, pp. 27–37.
5. B. Satzger et al., "Esc: Towards an Elastic Stream Computing Platform for the Cloud," *IEEE Int'l Conf. Cloud Computing*, 2011, pp. 348–355.

storage capabilities next to the sensor; on clusters with fixed computational resources; or on a cloud infrastructure that provides unlimited computational resources and unlimited data storage.

Besides the volatile resource requirements, it's also important to consider the communication channels between the different nodes' locations, as well as relevant SLAs (for example, the maximum processing time for streaming data or the ratio of discarded streaming data). These different aspects form a nondeterministic polynomial time (NP)-complete problem for an optimal deployment solution in terms of costs, performance, and SLAs. Along with the optimal initial deployment, it's important to investigate robust migration strategies for processing nodes. Because the IoT landscape is constantly evolving, we might need to migrate processing nodes from one geographical location to another during runtime. This requires a sophisticated migration strategy to ensure a consistent state for stateful stream processing nodes while maintaining continuous uptime and responsiveness.

Data Elasticity

Optimal and efficient access to the data from remote data repositories is required, but this efficiency gain must not reduce the streaming data's quality and availability. Because the data originates from geographically distributed as well as inconsistent or even conflicting data sources, we must develop methodologies for stream processing to consolidate the streaming data at runtime. These methodologies should be robust when data are noisy, uncertain, delayed, or (at worst) unavailable. Current advances in Semantic Web technologies represent one possible solution approach that could help define requirements on data and its representation. To achieve this, the level of description must be minimal to avoid overloading the stream processing system, which aims to manipulate light objects with only minimal operational overhead.

Based on current state-of-the-art stream processing and the future challenges we anticipate with the IoT, we identified several open areas for research in elastic stream processing for distributed environments. Although there's room for improvement, elastic stream processing can handle the challenges issued by the IoT to realize smart systems.

Acknowledgments

This article is supported by Vienna University of Technology research funds. This work is partially supported by the Commission of the European Union within the SIMPLI-CITY FP7-ICT project (grant agreement 318201).

References

1. M. Cherniack et al., "Scalable Distributed Stream Processing," *Proc. Conf. Innovative Data Systems Research*, 2003, vol. 3, pp. 258–268.
2. S. Acharya et al., "The Aqua Approximate Query Answering System," *ACM Sigmod Record*, 1999, vol. 28, no. 2, pp. 574–576.
3. G. Copil et al., "SYBL: An Extensible Language for Controlling Elasticity in Cloud Applications," *Proc. 13th Int'l Symp. Cluster, Cloud, and Grid Computing*, 2013, pp. 112–119.
4. M. Armbrust et al., "Above the Clouds: A Berkeley View of Cloud Computing," *Comm. ACM,* 2010, vol. 53, no. 4, pp. 50–58.

Christoph Hochreiner is a PhD student at the Distributed System Group at the TU Wien, Austria. His research interests cover the whole spectrum of cloud computing, specifically stream processing systems, service monitoring, and security implications. Hochreiner has an MSc in software engineering and Internet computing from the Vienna University of Technology. Contact him at c.hochreiner@infosys.tuwien.ac.at; www.infosys.tuwien.ac.at/staff/hochreiner/.

Stefan Schulte is an assistant professor at the Distributed Systems Group at TU Wien and the project manager of the ongoing European Union's Seventh Framework Programme project SIMPLI-CITY – The Road User Information System of the Future (http://www.simpli-city.eu). His research interests span the areas of cloud computing and service-oriented computing, with a special focus on aspects of quality of service. Schulte has a PhD in Web Service discovery from the Technical University of Darmstadt. Contact him at s.schulte@infosys.tuwien.ac.at; www.infosys.tuwien.ac.at/staff/sschulte/.

Schahram Dustdar is a full professor of computer science and he heads the Distributed Systems Group at TU Wien. His work focuses on Internet technologies. Dustdar is a member of the Academy Europeana, an ACM Distinguished Scientist, and recipient of the IBM Faculty Award 2012. Contact him at dustdar@dsg.tuwien.ac.at; dsg.tuwien.ac.at/.

Freddy Lecue is a research scientist and lead investigator in large-scale reasoning at IBM Research–Ireland. His research focuses on AI, knowledge representation and reasoning, service computing with a focus on Semantic Web technologies, and (hybrid) reasoning from various domains. Lecue has a PhD in computer science from Ecole des Mines de Saint-Etienne, France. Contact him at freddy.lecue@ie.ibm.com; http://researcher.watson.ibm.com/researcher/view.php?person=ie-freddy.lecue.