Orchestrating Apache NiFi/MiNiFi within a Spatial Data Pipeline

Chase Carthen*, Araam Zaremehrjardi*, Vinh Le*,
Carlos Cardillo‡, Scotty Strachan†, Alireza Tavakkoli*, Frederick C. Harris* Jr., Sergiu M. Dascalu*
**Computer Science and Engineering*,†*System Computing Services*,‡*Nevada Center for Applied Research*
‡**University of Nevada, Reno*,†*Nevada System of Higher Education*
Reno, Nevada
Email: {ccarthen,azaremehrjardi,vle,tavakkol,ccardillo}@unr.edu,
{fred.harris,dascalus}@cse.unr.edu,sstrachan@nshe.nevada.edu

***Abstract*—In many smart city projects, a common choice to capture spatial information is the inclusion of LiDAR data, but this decision will often invoke severe growing pains within the existing infrastructure. In this paper, we introduce a data pipeline that orchestrates Apache NiFi (NiFi), Apache MiNiFi (MiNiFi), and several other tools as an automated solution in order to relay and archive LiDAR data captured by deployed edge devices. The LiDAR sensors utilized within this workflow are Velodyne Ultra Pucks sensors that capture at a rate of 10 frames per second and produces 6-7 GB packet capture (PCAP) files per hour. By both compressing the file after capturing it and compressing the file in real-time, we discovered that gzip produced a file of 5 GB and saved about 5 minutes in transmission time to NiFi, as well as saving considerable CPU time when compressing the file in real-time. Alternatively, we chose XZ as the compression algorithm for the ingestion of LiDAR data onto an institution compute cluster due to its high compression ratio. In order to evaluate the capabilities of our system design, the features of this data pipeline were compared against existing third-party services, namely Globus and RSync.**

***Index Terms*—nifi, minifi, iot, data transfer, big data, smart city, PCAP, LiDAR, data pipeline, edge computing**

I. Introduction

As cities begin employing more and more complex sensing devices to either conduct traffic analysis or provide a measure of infrastructure, creating a system for data transferal becomes a crucial challenge. For smart city projects, spatial information such as Light Detection and Ranging (LiDAR) is especially a concern. Due to the massive amount of data generated by LiDAR point clouds, data collection and transferal from edge device to central repository tends to suffer from bottle-necking issues, such as low throughput networking, high latency, and packet-loss. These constraints must be considered as most cities in the United States may have difficulty placing fiber optic infrastructure in their cities [1].

As part of ongoing smart city developments in the city of Reno, Nevada, the work presented within this paper involves a 100 mbps fiber network provided by the City of Reno. While this network was deployed to specifically address the Cyberinfrastructure needs within the city of Reno, this called for the development of a Software Data Pipeline (SDP) that could enable reliable data transformation, transferal, and logging between edge computers and the fog computing network.

In this work, we developed the before-mentioned SDP, which uses NiFi/MiNiFi to facilitate the movement of LiDAR data generated at edge computing location placed around the city of Reno. This data is relayed to the fog computing network located on the University of Nevada, Reno (UNR) and then finally piped towards its final destination, UNR's Pronghorn High Performance Computing Cluster (HPC) for archival storage. The software on the edge environments use Docker Compose with MiNiFi to hook into the NiFi-based data pipeline in which the LiDAR point-clouds are compressed and then transmitted off. The software within the UNR Data Center uses Kubernetes to scale up NiFi hosts and receive the LiDAR point clouds which are then processed for storage.

The presented solution does offer insights to those interested in establishing a scalable pipeline for spatial data collection within smart city infrastructure [2]. With the increasing interest in smart city development, our approach fulfills a template so that other cities with similar network infrastructure may easily incorporate LiDAR data collection as part of their normal workflow. Due to the versatility of LiDAR data, LiDAR collection presents more opportunities for cities to better utilize big data methodologies for effective planning or the establishment of new data-driven solutions [3], [4].

As a form of evaluation for this SDP, we conducted an analysis of different compression algorithms, compared our approach with the present network bandwidth, and finally performed a feature comparison with major established third-party services, RSync [5] and Globus [6]. In addition, our solution has been tested by gathering the bandwidth usage, resource usage on edge devices, and recording time for message transfer. To elaborate, this involved testing different compression methods in terms of resource usage, average CPU usage, average memory usage, total duration time, and size of messages. As part of the feature comparison, RSync and Globus will be compared against our solution for basic functionality of data transmit and receive, load balancing, parallel streaming support, the customization of data flow, and file verification.

The remaining of this paper is structured as follows: Section II presents background information of the technologies explored and used by our approach, Section III describes

the design of the software data pipeline with considerations and expected requirements of the data pipeline, Section IV details the resulting implementation of the planned design and data flow, and Section V presents the overall performance evaluation of the software data pipeline with benchmarks and comparisons of other methods, and Section VI discusses possible uses of the data pipeline and outlines future work to extend its functionality.

II. Background and Related Works

With the increasing interest in both cloud and fog computing, different approaches have been explored to facilitate streams of data that require high throughput, intense bandwidth usage, and consistent access across different network scenarios. In general, the very nature of these data streams present certain difficulties to system architects when designing the software facilitating the stream and may require advanced big data techniques. In response to this, Software Data Pipelines (SDPs) are often presented as solutions to abstracting data streams by utilizing either custom software or preexisting suites of third-party tools. One such SDP takes a different approach than our solution, in that this data pipeline explored a combination of MQTT and Apache Kafka for processing data streams originating from industrial IoT devices [7], [8]. Another similar approach involved the usage of NiFi and MiNiFi to process different types of spatial information captured from Twitter streams and provide an accompanying sentiment analysis service [9], [10].

As mentioned, a major portion of our solution relies on NiFi and MiNiFi. Together NiFi and MiNiFi are an especially viable solution for SDPs that enables APIs to build data transformers, loggers, and other data flow measures. NiFi is composed of components called ”Processors” and the data passed between these processors, dubbed ”Flow Files”, which contain the data being transferred and additional metadata pertaining to the transfer. These processors used in a NiFi pipeline allow one to create, remove, modify, or inspect the contents of a flow file. Additionally, NiFi/MiNiFi allows users to create their own processors, which opens the door for greater control and manipulation of the data within a pipeline. NiFi operates with data producers and data consumers, dubbed ”Agents” and, alongside the flow files and processors, represent the abstract building blocks to a SDP within NiFi. However, NiFi in itself has the capability of over-bloating a system with tools that are not necessary for remote systems with constrained resources. While NiFi offers the full-suite of tools for building data pipelines, MiNiFi addresses the previous concerns by providing a bare-bones version made to run on resource-constrained edge devices and relay back information to a NiFi-based pipeline. This makes NiFi and MiNiFi ideal to be used for scenarios in which a SDP would be developed to collect spatial information and then process that data within a fog or cloud based environment.

As part of the work presented in this paper, we used NiFi [11] and MiNiFi [12] to create a SDP that allows for the collection and storage of LiDAR data being gathered from sensors installed within in the Virginia St. University corridor of Reno, Nevada. These LiDAR sensors are a series of Velodyne Ultra Pucks, and each are capable of creating a 360 degree point cloud of the intersections [13]. Each intersection within this space has two LiDAR sensors installed diagonally northeast and southwest in order to establish a consistent setup. The Ultra Puck sensors each produce approximately 300,000 points per second which equates to about 6 GBs of data produced per hour. To account for this big data problem in near real-time, researchers started using Apache Cassandra and Spark to compute digital terrain maps or other data stream processing frameworks [14], [15]. However, our approach would be more similar to a system developed by Michael et al. where the authors created a method for handling different bandwidths to a compute server [16]. However, our system does not perform any computation with the underlying data from the LiDAR.

As part of our evaluation of the work presented in this paper, Globus was chosen as a suitable system to compare the features of our SDP against, due to its popularity within the domain. Globus is a PaaS (Platform as a Service) created by the University of Chicago, now operating as a non-profit service, used to store and transmit data. The platform allows developers to use either a software development kit or REST APIs to create Flows within Globus. Flows are the basic-building blocks to generate SDPs within the Globus system, with Action Providers allowing users to extend the data pipeline. Action Providers open the door for some customization of the basic data pipeline used to transfer files in Globus. Globus provides a wide berth of features that include the ability to not only send and receive data, but the ability to load balance, transmit data in parallel streams, and validate incoming/outgoing files.

In the same lines as Globus, RSync was also chosen as a system for comparison, due to its incredible popularity and usage within the domain. RSync is a command-line utility that prioritizes performance over usability in order to transfer files between a source and destination host, while offering features for advanced customization of an data pipeline built using the technology. This utility, while offering limited functionality, still possesses the ability to robustly send and receive data, as well as provide a decent measure of file verification. This makes it ideal for creating data pipelines that continuously stream a directory from the source host to the destination host. However, if the needs of a data pipeline were to evolve to include different data processing, logging, or transformation measures, this would entail stringing multiple RSync applications together to create a more customized data pipeline.

III. Design of The Data Pipeline

The data pipeline was designed with the consideration that the raw LiDAR data would be stored into some archive where researchers may use the data for post analysis. In order to ensure that this process was possible, we had to make sure that the software pipeline that we choose is both scalable and conserves resources, such as bandwidth, CPU, and storage.

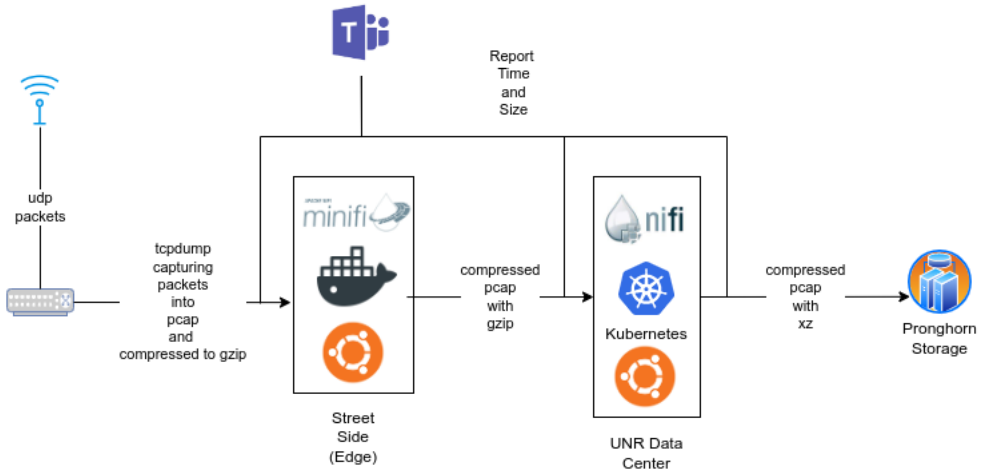

Fig. 1. High level diagram of the proposed pipeline architecture showing the flow of the PCAP from the edge to Pronghorn.

The data pipeline consists of several different components running on three different hosts. These three different hosts are: the edge computers at the street side, the UNR data center serving as the centralized hub, and the UNR Pronghorn HPC Cluster. Below are some requirements that we had in mind in designing this data pipeline:

- The data pipeline shall automate the process of sending LiDAR data as compressed files with set name schema.
- The data pipeline shall compress data in order to minimize the amount of network bandwidth and storage used at the edge and on pronghorn storage.
- The data pipeline shall report the amount of time taken in order to save.
- The data pipeline shall use compression algorithms to minimize the amount of CPU used at the edge and maximize the amount of compression at the pronghorn storage.
- The data shall be archived on the Pronghorn HPC Cluster,
- The data pipeline shall be scalable and easily orchestrated.

We designed the data pipeline to keep these requirements in mind. We decided to choose a store and forward approach as our requirements included only archival. The store and forward approach required us to take into consideration how much space was available on the device and how much bandwidth was available on the network. In this case, the design was based on a 100 mbps network. This network limited how much data could be sent as it was shared across six different intersections from the edge to the the data center so we had to figure out ways cut down on bandwidth by compressing the data. Even with an expansion of network bandwidth capabilities, we still wanted to constrain how much data is sent and have some form of quality of service in place in order to allow further applications to also run on our pipeline.

Fig. 1 shows the overall pipeline that we designed along with the software used to make it possible, together with the reporting of time and size of data along the pipeline. Data is generated by a Velodyne Ultra Puck sensor emitting UDP packets that are then acquired by an edge computer connected to a city street light. The UDP packets are captured using tcpdump and stored into a PCAP file. The PCAP file with gzip at this point is either compressed at a later date or compressed in real-time, then sent from the edge to the data center. At the data center, the compressed PCAP file is uncompressed from a gzip file to a XZ file and then sent to Pronghorn for archival storage.

At the edge, Docker Compose was used to orchestrate the setup of tcpdump and MiNiFi. Docker Compose was chosen as the setup of this infrastructure because it could be easily replicated on any machine that has Docker Compose installed. Docker Compose allows for rapid modification of the configurations of MiNiFi and tcpdump to try different versions of the software and different configurations without changing the underlying operating system. It also handles the setup of networking between any software that is used.

At the data center, Kubernetes was used to orchestrate the setup of NiFi and all the components that it needs to run in a cluster setup. Kubernetes was chosen because it can setup or scale many different types of software across multiple machines with ease. The cluster version of NiFi was selected to allow for the load balancing features to be used and enabled NiFi to scale for other future projects.

IV. Implementation of The Data Pipeline

At the edge, our setup included a Cincoze DS-1200 with Ubuntu 18.04 installed, with a total 16 GB for RAM and Intel CPU i7-8700T. At the data center, our configuration consisted of an 8-node kubernetes cluster spread across two four-unit machines. Those machines were a SuperMicro SYS-6029TP-HTR and X11DPT-PS. The Kubernetes cluster was setup to handle the workflow of this project and other projects on campus. Additionally, a Supermicro X11DPH-T with 20 TB of allocated storage was put in place to serve as an intermediate storage for the 8-node Kubernetes cluster. This intermediate

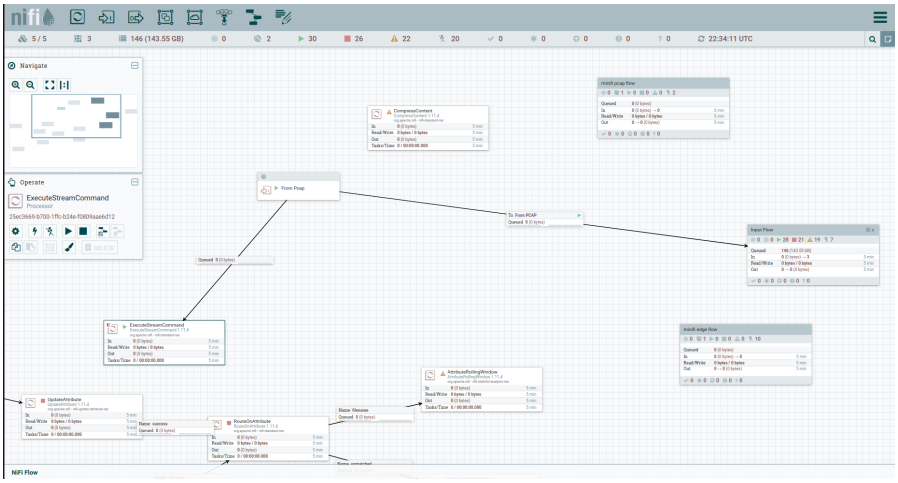

Fig. 2. Screenshot of NiFi showing a portion of the data flow from the edge to the data center.

storage was used as a staging area before sending it off to Pronghorn. All 9 of these machines sit in the same rack at the data center located on the UNR campus. Each machine also used Ubuntu 18.04 for the operating system. We used Rancher RKE 1.0 for the Kubernetes distribution.

NiFi, MiNiFi, and tcpdump were all placed into Docker containers. Both NiFi and MiNiFI were configured with yaml files to set the settings of the software and data pipeline. The tcpdump container was designed and constructed in such a way that the collection interval and the name of the file can be changed to include descriptive labeling such as: intersection names, what corner the LiDAR sensor is on, and the municipal location of the sensor. In regards to the schedules we decided to have the tcpdump container collect every hour and roll to the next file to collect in another hour.

At five different intersections that make up the Virginia St. Corridor, the two Ultra Puck sensors collect at a rate of 10 revolutions per second. This PCAP file after being fully collected in an hour is moved into a directory where MiNiFi will place the file into a queue to be sent off to NiFi. When NiFI receives the file from the edge, it will place the file into a storage server within the UNR data center. Inside of NiFi, it will queue any files that land on the storage server and start to process the PCAP file by compressing it into a XZ file. Before it is compressed into XZ, it will check to see if the file has already been compressed with gzip and if it has been compressed, then it will decompress the file. After the file has been compressed with XZ, it will finally transfer the file off to Pronghorn. During this whole process to keep track of time, we have NiFi send completion time statistics to a Microsoft Teams channel setup between the Edge and NiFi, as well as between the storage server at the UNR Data Center and Pronghorn.

Fig. 2 shows a screenshot of NiFi a part of our workflow. A user of NiFi can specify their workflow in the user interface. NiFi was chosen due to the ease of use and being able to specify data flows with a GUI. This particular screenshot demonstrates where NiFi grabs data from MiNiFi and stores it onto a storage chassis in the UNR data center, while statistics about the whole process are sent off to Microsoft Teams.

Our approach to a data pipeline for transferring raw LiDar utilizes NiFi's and MiNiFi's Site-to-Site protocol, which in turn makes use of the RAW transport protocol setting. NiFi's Site-to-Site Protocol allows us to use any number of MiNiFi instances to connect to our clustered instance of NiFi. This allow us to scale MiNiFi running at multiple different intersections. We designed the MiNiFi flow within NiFi and then exported it for the deployment at the edge.

We compressed the PCAP file in two different ways at the edge. The first method involved compressing the file after is has been fully captured. The second method was compressing the file in real-time. The first method is the easiest to implement and provides a good comparison for the second method. The second method was implemented because it saves the time of compressing the file afterword, but at the cost of extra computation being during recording. We tested the lz4, bzip, xz, gzip, and zstd compression algorithms. We chose gzip as the compression algorithm between the edge and data center communication. For archival purposes, we chose xz as the compression algorithm at Pronghorn.

V. Results and Discussion

A subset of results were collected from Microsoft Teams and used to find out how long the overall transfer time takes. We found that NiFi and MiNiFi both send information at the maximum network bandwidth supported by the 100 mbps network. Table I shows that the file size of the compression shaves off about 2 GB and saves about 3-4 minutes of transmission time. Our results show that NiFi adds about 10-12 seconds on transfer time overhead while under normal conditions. This test was conducted to ensure that the implementation of this software does indeed use the file network with no large bottleneck on part by the software itself.

Both methods of compression from Section IV were compared and measured using the ps command from Linux. Both

TABLE I
THE BANDWIDTH USAGE OF ORIGINAL AND COMPRESSED DATA.

Method	Size	Bandwidth	Time
Original	7 GB	11 MB/sec	10-12 min.
Original with Compression	5 GB	11 MB/sec	7-8 min.

methods were ran on two different PCAP files that were recorded for one hour. Table II demonstrates the results for the first method where a file is compressed after being collected from LiDAR sensors. Table III demonstrates the results for the other method were the file is compressed as it is collected from the LiDAR sensor. In Table III, the duration within the table represents the amount of time the program actually ran on the CPU, while the PCAP file was captured for one hour as reported by the PS command. All compression algorithms included in this experiment were used with their lowest and fastest setting.

TABLE II
NON-REAL-TIME COMPARISON

Compress Method	Average CPU%	Average% Mem.	Total Duration	Size (GB)	Compression Ratio
lz4	96.12	0.097	0:30.0	3.9	0.19
bzip2	99.49	0	14:32.0	3.2	0.33
xz	99.94	0.010	32:53.0	1.9	0.6
gzip	99.25	0	03:56.0	3.3	0.31
lzma	99.92	0.010	32:28.0	1.9	0.6
zstd	98.84	0	01:09.0	3.4	0.29
Original	-	-	-	4.8	-

Comparing the two tables, both xz and lzma have the best compression ratio, but take the most time in comparison to the other compression algorithms. Examining the second method in comparison to the first method, the second method doesn't nearly use as much CPU when a file is being recorded. Looking at gzip and lz4 for the second method, both compress the original file to 3.9 GB and 5.2 GB from 6.4 GB. Out of these two compression algorithms gzip compresses better and only at a slightly higher CPU usage. Based on these results, we chose gzip to be the compression algorithm for the edge to data center communication. XZ was chosen to be the compression algorithm due to its high compression ratio for storage on Pronghorn.

TABLE III
REAL-TIME COMPARISON

Compress Method	Average CPU%	Average% Mem.	Total Duration	Size (GB)	Compression Ratio
lz4	0	0	1:06.0	5.2	0.19
bzip	27.14	0	20:01.0	3.4	0.47
xz	0.010	0.010	38:01.0	2.5	0.61
gzip	0.0006	0	07:40.0	3.9	0.39
lzma	9.86	0	38:31.0	2.5	0.61
zstd	1.94	0	03:49.0	4.4	0.31
Original	-	-	-	6.4	-

Table IV shows a feature comparison across Globus, RSync, and our approach with NiFi and MiNiFi. Globus makes use of GridFTP to send data from one data source to another. Globus is typically used for larger files and supports parallel network streams when sending files. This allow for Globus to send files much faster in comparison to RSync and our approach. All three approaches support performing file verification or some form of check summing, but our approach would have to specifically implement it within the data flow of NiFi and MiNiFi. Globus allows for universities to scale up their end points and underneath the hood of their software uses cloud software to handle sending data between two different data sources. NiFi is able to be scaled up due to being able to be clustered with the help of Zookeeper. RSync only supports a straight end to end connection from one host to another host and does not perform load balancing or scalability. The best that could be used with RSync is starting up multiple instances of RSync.

TABLE IV
A FEATURE COMPARISON OF OUR APPROACH VS GLOBUS AND RSYNC.

Features	Globus	RSync	Our Approach
Send/Receive Data	X	X	X
Load Balancing	X		X
Send Data in Parallel Streams	X		
Customizable Data Flow			X
File Verification	X	X	X

Both RSync and Globus are not able to support custom data flows like NiFi or MiNiFi. As explained before NiFi allows for the users to send their data to many different types of options like a database, another NiFi, to a web service, and many others. This flexibility allows for us to potentially send the data to other sources for instance to cold storage or other collaborators who want a live copy of the data. We could also perform analytics on the data as it moves through NiFi. Both Globus and RSync are good for sending data from one source to another and have some analytics, but they lack the flexibility that NiFi gives with creating data flows inside a user interface. However, NiFi may take some time to setup and lacks some of the ease of use of Globus's user interface to send files within their platform. RSync is readily available on Linux and can be utilized by installing it as a package. While NiFi may take some setup effort and require some specific configuration, the ability to alter the data flow with a user interface makes it easier to visualize the flow of the data.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we discovered that using NiFi and MiNiFi produced a promising solution for transferring LiDAR data. Additionally, XZ was found to be the best compression algorithm for archiving onto the UNR's HPC cluster due to its high compression ratio. Furthermore, compressing the PCAP file in real-time only used a minimal amount of CPU power in comparison to recording the PCAP file and then compressing it afterward. This compressing style shaved off about 5 minutes in transmission time and saved about 2 GB in file size. Through our feature comparison, we discovered that our approach covers a significant breadth of service among similar systems, but still lacks certain advanced features, such as

enabling parallel streams. Additionally, under this evaluation, we also found that our approach had greater flexibility and ease of use due to the user interface provided by NiFi.

As part of our future work, we plan to expand this method by exploring new avenues, such as sending the PCAP to a web service to convert the raw data within the PCAP into point cloud format. This would then be stored in a database with NiFi. NiFi allows for us to quickly try out different quality control (QC) and quality assurance (QA) implementations. These implementations could be applied for example when the file comes from the edge to the data center or when the file is placed into the archive at the data center. Finally, we could potentially adapt the approach presented in this paper with additional devices integrated along the Virginia St. Corridor, such as video cameras or various time-series sensors.

ACKNOWLEDGMENT

The authors would like to acknowledge the support of Research & Innovation and the Cyberinfrastructure Team in the Office of Information Technology at the University of Nevada, Reno for facilitation and access to the Pronghorn High-Performance Computing Cluster. The authors would like to acknowledge the City of Reno. This material is based in part upon work supported by Washoe County Regional Transportation Commission(RTC) under grant number AWD-01-00002406. It is also based in part upon work supported by the National Science Foundation under grants OAC-2209806, OIA-2019609, and OIA-2148788. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Washoe County RTC or The National Science Foundation.

REFERENCES

[1] T. Cooper. "Municipal broadband 2022: Barriers remain an issue in 17 states." (Oct. 2022), [Online]. Available: https://broadbandnow.com/report/municipal-broadband-roadblocks/ (visited on 03/09/2023).

[2] M. Duygan, M. Fischer, R. Pärli, and K. Ingold, "Where do smart cities grow? the spatial and socio-economic configurations of smart city development," *Sustainable Cities and Society*, vol. 77, p. 103 578, 2022, ISSN: 2210-6707. DOI: https://doi.org/10.1016/j.scs.2021.103578. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S221067072100843X.

[3] S. McCrae and A. Zakhor, "3d object detection for autonomous driving using temporal lidar data," in *2020 IEEE International Conference on Image Processing (ICIP)*, 2020, pp. 2661–2665. DOI: 10.1109/ICIP40778.2020.9191134.

[4] J. Zhao, H. Xu, H. Liu, J. Wu, Y. Zheng, and D. Wu, "Detection and tracking of pedestrians and vehicles using roadside lidar sensors," *Transportation Research Part C: Emerging Technologies*, vol. 100, pp. 68–87, 2019, ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2019.01.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X19300282.

[5] S. Org. "Rsync." (), [Online]. Available: https://rsync.samba.org/ (visited on 03/09/2023).

[6] I. Foster, R. Kettimuthu, S. Martin, *et al.*, "Campus bridging made easy via globus services," in *XSEDE '12*, ACM, Chicago, IL: ACM, Jul. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2335847.

[7] M. Helu, T. Sprock, D. Hartenstine, R. Venketesh, and W. Sobel, "Scalable data pipeline architecture to support the industrial internet of things," *CIRP Annals*, vol. 69, no. 1, pp. 385–388, 2020.

[8] T. P. Raptis, C. Cicconetti, M. Falelakis, G. Kalogiannis, T. Kanellos, and T. P. Lobo, "Engineering resource-efficient data management for smart cities with apache kafka," *Future Internet*, vol. 15, no. 2, p. 43, 2023.

[9] S.-S. Kim, W.-R. Lee, and J.-H. Go, "A study on utilization of spatial information in heterogeneous system based on apache nifi," in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, 2019, pp. 1117–1119. DOI: 10.1109/ICTC46691.2019.8939734.

[10] A. Pandya, P. Kostakos, H. Mehmood, *et al.*, "Privacy preserving sentiment analysis on multiple edge data streams with apache nifi," in *2019 European Intelligence and Security Informatics Conference (EISIC)*, 2019, pp. 130–133. DOI: 10.1109/EISIC49498.2019.9108851.

[11] A. S. Foundation. "Apache nifi." (2023), [Online]. Available: https://nifi.apache.org/ (visited on 03/10/2023).

[12] A. S. Foundation. "Apache minifi." (2023), [Online]. Available: https://nifi.apache.org/minifi/ (visited on 03/10/2023).

[13] V. L. Inc. "Ultra puck surround view lidar sensor." (), [Online]. Available: https://velodynelidar.com/products/ultra-puck/ (visited on 03/09/2023).

[14] D. Deibe, M. Amor, and R. Doallo, "Big data geospatial processing for massive aerial lidar datasets," *Remote Sensing*, vol. 12, no. 4, 2020, ISSN: 2072-4292. DOI: 10.3390/rs12040719. [Online]. Available: https://www.mdpi.com/2072-4292/12/4/719.

[15] H. Isah, T. Abughofa, S. Mahfuz, D. Ajerla, F. Zulkernine, and S. Khan, "A survey of distributed data stream processing frameworks," *IEEE Access*, vol. 7, pp. 154 300–154 316, 2019.

[16] S. Oh, J.-H. You, A. Eskandarian, and Y.-K. Kim, "Accurate alignment inspection system for low-resolution automotive lidar," *IEEE Sensors Journal*, vol. 21, no. 10, pp. 11 961–11 968, 2021.