Scheduling Stream Processing Tasks on Geo-Distributed Heterogeneous Resources

Gerrit Janßen, Ilya Verbitskiy, Thomas Renner, and Lauritz Thamsen
Technische Universität Berlin, Germany
{firstname.lastname}@tu-berlin.de

***Abstract*—Low-latency processing of data streams from distributed sensors is becoming increasingly important for a growing number of IoT applications. In these environments sensor data collected at the edge of the network is typically transmitted in a number of hops: from devices to intermediate resources to clusters of cloud resources. Scheduling processing tasks of dataflow jobs on all the resources of these environments can significantly reduce application latencies and network congestion. However, for this schedulers need to take the heterogeneity of processing resources and network topologies into account.**

This paper examines multiple methods for scheduling distributed dataflow tasks on geo-distributed, heterogeneous resources. For this, we developed an optimization function that incorporates the latencies, bandwidths, and computational resources of heterogeneous topologies. We evaluated the different placement methods in a virtual geo-distributed and heterogeneous environment with an IoT application. Our results show that metaheuristic methods that take service quality metrics into account can find significantly better placements than methods that only take topologies into account, with latencies reduced by almost 50%.

***Index Terms*—Task Scheduling, Operator Placement, Stream Processing, Quality of Service, Resource Management**

I. Introduction

The Internet of Things (IoT) is transforming a growing number of areas, including for example manufacturing as Industry 4.0 [1], municipalities into Smart Cities [2], and ordinary houses becoming Smart Homes [3]. In many IoT applications, devices equipped with sensors continuously record data that is then transmitted and analyzed. Often enough these applications are executed in geo-distributed computing environments. That is, sensor data is recorded far away from data centers such as in people's houses. Yet, first transmitting the data from the sources to homogeneous cluster resources leads to significant response times for streaming applications. Addressing this, the ideas of Edge and Fog Computing have become important: To reduce latencies and network load, the available resources close to data sources and on network paths to central cluster resources are increasingly used to execute parts of analytics pipelines [4, 5, 6].

At the same time, distributed dataflow frameworks like MapReduce [7], Spark [8], and Flink [9] have been developed for efficiently processing large volumes of data using commodity clusters. Increasingly these systems also allow and are used for scalable processing of data streams [10, 9, 11]. However, the frameworks currently still typically expect to run on homogeneous cluster resources. That is, neither the heterogeneity of resources nor the topology of the network is taken into account when jobs are scheduled onto resources.

Given the increasingly distributed and heterogeneous computing environments of IoT applications of today, the goal becomes to effectively schedule dataflow jobs based on the varying capacities of nodes as well as network properties like latencies and bandwidths to fulfill performance requirements. Towards this goal, this paper examines multiple placement strategies that make use of Quality of Service (QoS) metrics as well as network and job topologies. The first class of methods we examine here are *metaheuristic search algorithms* making use of the QoS metrics. These algorithms search for optimal placements by stepwise exploration of the neighborhood of an initial solution, using an optimization function for assessing the potential solutions. The second class of methods we present are those that find optimized placements solely based on the structure of the network and the streaming job. These *constructive algorithms* create solutions by beginning with an empty solution and adding components iteratively until a valid solution emerges.

For our evaluation, we extended the distributed dataflow system Apache Flink. For this, we added the possibility for users to express the resource requirements of tasks as well as the capabilities of worker nodes. What is more, we extended Flink's scheduling workflow to use our scheduler on a job submission, which then makes use of the data collected.

In summary, our contributions are as follows:

- The definition of QoS metrics regarding bandwidth, latency, and computational resources, which are combined into an optimization function.
- The implementation of metaheuristic algorithms using our optimization function and constructive methods matching the graph structures for optimal task placements.
- An evaluation of all placement methods in a simulated geo-distributed heterogeneous compute environment using GNS3, our extended Flink system, and an IoT benchmark application.

The remainder of the paper is structured as follows. Section II discusses related work for placing tasks of distributed dataflows in heterogeneous environments. Section III explains our QoS metrics, our optimization function, and constructive methods. Section IV presents our evaluation. Section V concludes this paper and also presents ideas for future work.

II. Related Work

This section presents an overview of work towards efficiently processing data from geo-distributed locations with distributed dataflow systems and heterogeneous heterogenous resources.

Iridium [12] is an approach based on Apache Spark that combines task and data placement to achieve low query response times. It uses an online heuristic to re-distribute datasets over different sites before the execution of a query takes place. To reduce network bottlenecks during the query, execution tasks are placed according to network conditions. In Iridium a distinction between reduce and join tasks of a MapReduce job is made and the placement is done by solving a Linear Program (LP). The LP is used to avoid bottlenecks and to minimize the overall bandwidth usage.

JetStream [13] is a system that allows real-time analysis of large and widely distributed evolving datasets, such as log or audiovisual data. It combines task and data placement strategies. The system uses storage at edge locations to save data for later usage and it places tasks according to data locations to optimize resource usage. JetStream uses a congestion monitor to place operators according to current network characteristics. Additionally, it provides an adaptive degradation functionality to realize quality tradeoffs, for instance in low-bandwidth environments.

Distributed Storm [14] is a scheduler extension for the Apache Storm framework [15] to realize an efficient task placement that satisfies QoS requirements. The work takes heterogeneous computing and network resources into account by formulating a general LP to place dependent tasks of dataflow applications. Heterogeneity is measured by QoS metrics: response time and availability. To solve this problem, they provide heuristics that optimize network-related QoS metrics, such as end-to-end latency, availability, and data exchange rates among nodes. The work is based on the metrics presented by Pietzuch et al. [16].

In our placement methods, we do not make a distinction between different operators as is done in Iridium. The heterogeneity of the operators in our approach derives from the dependencies of the tasks, the input/output rates as in Distributed Storm, and categories of operator requirements defined by users. In contrast to Iridium, we do not take the overall bandwidth usage into account, but instead the the overall bandwidth congestion. Our approach focuses on finding the optimal placement for dataflow tasks with regard to throughput and latencies by avoiding bandwidth congestion. The properties of the network and the dataflow jobs are used to realize an offline-scheduling before the execution of the dataflow job takes place. Thus, we neither consider changes of the network metrics nor changes of data rates as is done by JetStream. As with our approach, Distributed Storm uses QoS metrics to realize an optimized operator placement according to response times and availabilities. In our approach, availability is not considered, but the heterogeneity of compute resources and bandwidths between links are.

III. Approach

Our approach deals with the problem of placing tasks of a dataflow job with specific computational needs onto nodes with different computational resources in a network with different bandwidths and latencies between the nodes. Figure 1 depicts the general problem.

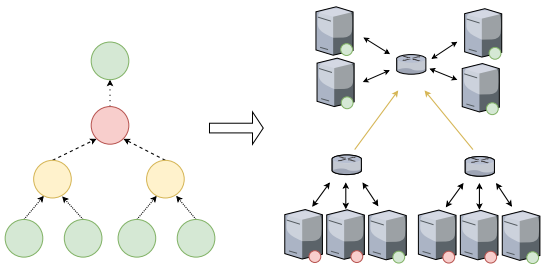

Fig. 1. We search for an efficient placement for tasks of a job graph (left) to resources in a network topology (right).

The job graph on the left specifies the flow of the data where a vertex represents a task and the edges show the direction of the data flow. Each task executes a user-defined function which produces data of varying size and at a varying speed. In addition, individual tasks might have different resource needs, which is indicated by the different colors of the vertices. In the job graph on the left, for example, green, yellow, and red mean low, medium, and high resource requirements, respectively.

The network topology on the right shows computation nodes that are interconnected with links. The nodes as well as the network links have different capabilities as shown by the colored badges on the nodes and the color of the links. In the example topology on the right, green, yellow, and red badges correspond to high, medium, and low resource capabilities, respectively. As for the link color in our example, a black arrow resembles a network connection without any substantial limitation whereas the yellow links represent network connections of limited bandwidth and increased latency.

Nodes in a topology can be direct or next level neighbors. We define direct neighbors as nodes that are connected to the same network switch and next level neighbors as nodes that are connected to a switch on the next hop. Based on these properties, we define leaf, intermediate and root nodes which build a directed graph along the topology. Leaf nodes are nodes, that are not a next level neighbor for any other node. Analogously, root nodes are nodes, which have no next level neighbors. Intermediate nodes are neither leaf nor root nodes.

For our approach, we assume such tree-like network topologies in which leaf and root nodes are designated to run source and sink tasks of a dataflow job, respectively. This way, we want to consider use cases where data is produced at sensor nodes and the results of a streaming dataflow job are then stored in the cloud.

A. QoS Metrics and Optimization Function

Before we proceed with the different metrics and optimizations function we need a more formal notion of task

placements. A placement is defined as a set of tuples which represent tasks of a dataflow job that are mapped to task slots on the nodes in the network topology. A valid schedule contains a mapping of every task of the job graph onto one single instance of the network topology, while source tasks are assigned to leaf nodes and sink tasks are assigned to root nodes. For a job graph G_t and a network topology G_i, a placement is thus a function $P_{G_t,G_i} : G_t \to G_i$. We assess the quality of a placement using three metrics. The following subsections describe them in more detail.

1) Response Time: Our consideration of response time (RT) is derived from [14]. It is considered as the time that is needed to transmit and process data items from source to sink tasks. Two dependent tasks might be placed on processing nodes that are connected by links and intermediate nodes. By default, we consider the response time of the dataflow as the longest time that is needed to transfer items from a source to a sink task. Alternatively, the user can specify a specific combination of a source and a sink task for which he wants to optimize the response time. For normalization purposes, we define an expectation for best (RT_{min}) and worst (RT_{max}) response times in our optimization function.

2) Bandwidth Congestion: For a task in a dataflow, we define its output as a combination of the data rate and the size of a single data item, that we call $\mathrm{DataIngestion}$. For source tasks, we assume an output rate R and an item size S. By combining these two quantities we can calculate the emitted amount of data of a source n per time unit as

$$\mathrm{DataIngestion}(n) = R \cdot S \ .$$

A processing task gets one or multiple data streams as input, does a modification and then emits one or more output streams. To take the processing into account, we define modification factors for the data rate (α) and the data item size (β). Hence, we define the data ingestion of a processing task t as

$$\mathrm{DataIngestion}_{\alpha,\beta}(t) = \alpha R \cdot \beta S \ .$$

The bandwidth congestion (BW) stands for the overloading of the network that might happen during the execution of the dataflow application. Links between nodes provide a certain amount of bandwidth that can be utilized to transfer data over this link. If the data that should be sent exceeds that capacity, a congestion occurs. The data then has to be stored in local caches as long as it cannot be sent over the link which leads to a processing delay. In queuing theory the effect is often called backpressure [17].

The bandwidth congestion is defined as follows. We calculate the required bandwidths of all directly dependent subtasks of the dataflow with the help of the previously introduced data ingestion which is defined as BW_{need} for every link of the topology. The topology itself provides information about available bandwidth resources between each pair of neighboring nodes that we define as BW_{avail}. Regarding the placement, the BW_{need} of a link is subtracted from it's BW_{avail}. The overall congestion is then calculated by the sum of all partial congestions that occur on the topology. Only congestion is penalized, not an overprovisioning of bandwidth. To ensure this, we reject positive terms of the sum that would imply an overprovisioning. More formally, let I the set of instances in the topology, then the bandwidth congestion is defined as

$$BW(I) = \sum_{(i_x,i_y)\in \mathrm{Neighbors}_I} \Big| \min\big(0, BW_{\mathrm{avail}}(i_x,i_y) - BW_{\mathrm{need}}(i_x,i_y)\big)\Big| \ .$$

3) Resource Fitting: As mentioned before, we have defined categories for the resource requirements of tasks and the resource provisioning of instances. The resource fitting (RF) formalizes how well an instance node can process a task with its requirements by mapping these categories to integer values and calculate a distance. The distances are then summed up, which results in a value for the whole placement P:

$$RF(P) = \sum_{(t,i)\in P} \Big| \min\big(0, r_{\mathrm{prov}}(i) - r_{\mathrm{req}}(t)\big)\Big| \ .$$

Note that, again, overprovisioning is not penalized. Our focus is to find a schedule that realizes the processing of the job as fast as possible and not to utilize the instances efficiently.

4) Optimization Function: As it is done in [14] we combine the QoS metrics for a placement P to define an objective function F:

$$F(P) = \frac{RT(P) - RT_{\min}}{RT_{\max} - RT_{\min}} + \frac{RF(P) - RF_{\min}}{RF_{\max} - RF_{\min}} + \frac{BW(P) - BW_{\min}}{BW_{\max} - BW_{\min}} \quad (1)$$

To adjust the QoS metrics into a common scale, the respective metrics has to be normalized by the min/max values.

B. Metaheuristic Methods

The steepest descent (SD) method and tabu search (TS) [18], start their optimization procedure with an initial valid placement. To create such a placement at the beginning, we assign source and sink tasks to suitable leaf and root nodes. Remaining tasks are placed to free slots in an arbitrary way. To explore the search space, we define two elementary steps. The *swap step* takes two assignments of the current solution and exchanges the placement of the tasks to the instances. The *switch step* moves a task to a slot that is not yet used for any assignment.

The neighborhood of a solution is defined as the set of solutions that result from applying a single elementary search step. In each round of the search, the neighborhood is determined and each possible solution is then evaluated using the objective function as defined in Equation (1). The solution with the lowest value is then chosen as the candidate for the best placement.

C. Constructive Methods

For the cases where no such QoS metrics are available, we use two simple methods that only require knowledge of the graph structures of the dataflow job and the topology: top down (TD) and highest level first (HLF).

A data transmission between tasks back to lower stages in the topology obviously leads to additional delays. Thus we can conclude that usually up-scheduling makes sense. This means, that in the scheduling of a successive task, nodes on the same or on a higher level are preferred. The idea behind the HLF approach is to take this into account. The creation of a scheduling begins at the source nodes where source tasks are assigned. Based on these assignments the subsequent tasks are scheduled on the nodes on the same level or on nodes that are at a higher level. A task can be scheduled on the same level when a free slot exists and the predecessor tasks are also scheduled on a direct neighbor in the topology. Otherwise, this task has to be scheduled on the next level instance. In addition, also the depths of the job graph and the network topology are considered. It is inspired by HLF approaches from multiprocessor scheduling [19]. Tasks of a taskgraph are ordered by their level and scheduled on nodes in this sequence to minimize the makespan. In homogeneous environments it does not matter where to schedule a task, only when. In our case, we also consider the levels of the task- and the topology graph and try to match the levels of nodes in both graphs. Thus, longer paths of the task graph are mapped to the longer paths of the topology graph.

While HLF starts the scheduling at source nodes, the TD method places the tasks from endpoint nodes in the direction of the source nodes. If enough task slots are available at the endpoint nodes, this method will schedule all intermediate and sink task on those nodes. To fulfill our initial assumptions source task are scheduled on source nodes nevertheless. If our endpoint nodes have high computation capacities (which we assume, because these nodes represents common cloud data centers), dataflow jobs with high computation requirements might benefit from this method.

IV. Evaluation

We evaluated our approach using a prototypical implementation in Apache Flink. This section provides a short overview over this implementation. Further, the section describes the testbed as well as the benchmarking application that was used for evaluating our approach. Finally, we discuss the evaluation results.

A. Prototype

We extended Flink's scheduler in order to integrate our scheduling approaches. Since our scheduler requires additional informations about the topology and the job, we extended Flink's user-facing API. For data source operators it is possible to specify the output item size and output rate. For subsequent data stream transformations, like map or filter operators, the output items size and and output rate modification coefficients can be set. For each operator in the job, also a minimum requirement for a resource (low, medium, high) can be defined.

To provide information about the topology and the capacities of the instances, the task manager configuration was extended by neighbor IDs as well as bandwidth, latency, and CPU limitations. These informations are parsed from the configuration file during the startup of the task managers, transmitted to the job manager node, and used by our scheduler implementation on a job request.

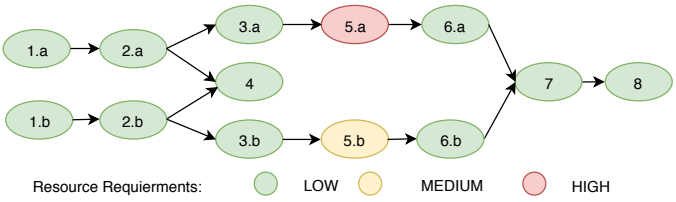

Fig. 2. Jobgraph of the Fire Detection benchmark job with different resource requirements.

a: Tasks for Tree Items b: Tasks for Location Items 1: Report Source
2: Report Splitter 3: Temperature Filter 4: Temperature Monitor
5: ImageProcessor 6: Result Filter 7: StreamConnector
8: FireAlertMonitor

B. Experiment Setup

For the evaluation, we used GNS3[1] that allows the emulation of complex networks using common router images and Docker containers that can be interconnected with each other. As Docker containers have limited network functionalities, GNS3 makes use of pipework[2] to connect containers to each other. This ensures that all egress traffic is routed through the defined network topology. To simulate the benchmark topology, multiple GNS3 instances were deployed on 9 machines. Each machine was equipped with a Intel Xeon CPU E3-1230 V2 3.30GHz, 16 GB RAM and a 1 GBit Ethernet NIC, running on CentOS. To connect multiple nodes and inject bandwidth limitations in our topologies we used Open vSwitch[3], which is a multilayer software switch that is widely used to connect virtual machines. The NetEm[4] appliance was used to simulated increased latencies between levels in the network topology. For CPU limitations, the option to constrain resources by the Docker Engine was used[5].

C. Benchmark

Our benchmark implements a fictional IoT use case that should detect fire in a forest environment with the help of temperature values and images, similar to applications described in [20, 21]. For this, areas of forests are continuously monitored by temperature sensors and cameras that generate reports with high resolution images for large locations (location reports) and low resolution images for single trees (tree reports). The reports consist of temperature values with fire and non-fire images that are processed by a fire recognition task in our dataflow. Figure 2 shows the job graph of the benchmark job.

The dataflow graph consists of source tasks (1a/b) that ingest reports, tasks to extract images from the reports (2a/b),

[1]https://www.gns3.com/, accessed: 13-09-2018
[2]https://github.com/jpetazzo/pipework, accessed: 13-09-2018
[3]https://www.openvswitch.org/, accessed: 02-11-2018
[4]https://docs.gns3.com/appliances/netem.html, accessed: 13-09-2018
[5]https://docs.docker.com/config/containers/resource_constraints/, accessed: 13-09-2018

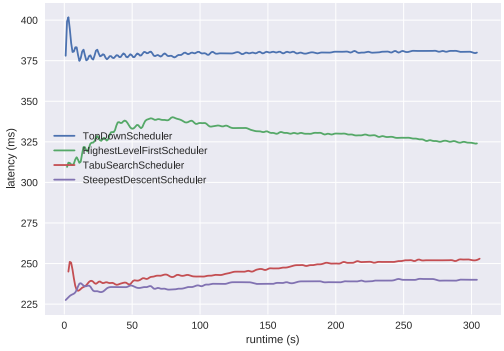

Fig. 3. End-To-End Latencies of Temperature Reports (Tree)

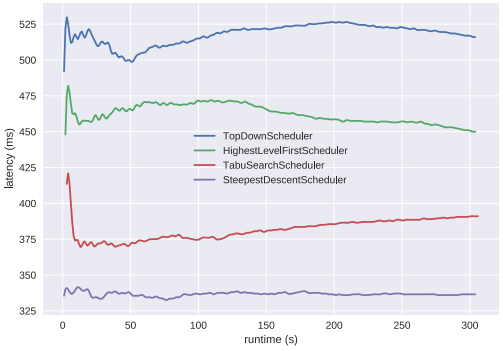

Fig. 5. End-To-End Latencies of Fire Alerts (Tree)

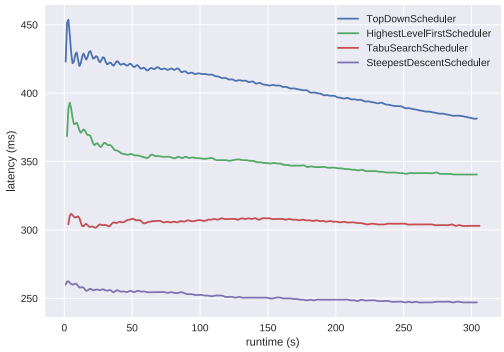

Fig. 4. End-To-End Latencies of Temperature Reports (Location)

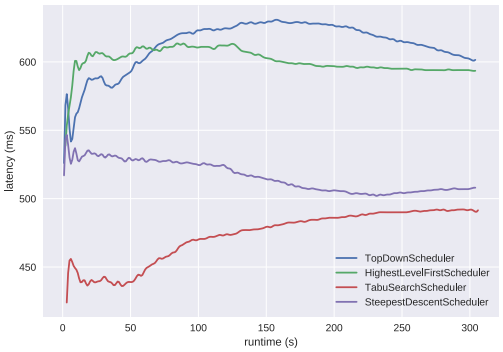

Fig. 6. End-to-End Latencies of Fire Alerts (Location)

filter and computation tasks (3a/b, 5a/b), and tasks that are used for monitoring (4,8). The ImageProcessor task perform a simple count of pixels and computes the share of red values in the image to detect a fire. The TemperatureMonitor gives a continuous view of the observed temperature, whereas the FireAlertMonitor represents an entity that might trigger an alarm if a fire has been detected.

Both, the amount of data that has to be transferred between tasks as well as the modifications of rates and sizes by the individual operators influence the required bandwidth. For the benchmark job, we used our extended Flink implementation to set these values. The output rate of the source operators was configured as the emission of a pre-defined number of reports per second. The data item size was determined by the size of a single report, which is known before the runtime. For task directly following the image processing (6a/b) we defined a data modification factor of 0.01 since the preceding tasks transform the incoming images to a single float value. For the data rates we defined modification factors according to the probability of a fire alert that could be set in our benchmark. Task requirements were usually set to low with the exception of the image processing tasks that were declared as medium for low resolution images and high for high resolution images.

The benchmark job was configured to ingest 100 tree reports and 25 location reports per second. To simulate sensors, the data was generated at the source tasks in an independent thread. During generation, a creation timestamp is set on each generated data record. The data records are then pushed to a waiting queue which is polled in the source task in a continuous manner.

For every scheduling method we executed the benchmark job for 5 minutes. Each scheduling method was allowed to run for at most 15 seconds. That is, the best schedule from the heuristic methods was chosen after this timeout. To have comparable plots, we have not taken the scheduling time into account.

We present the evaluation of the methods that were described in Section III on a flat topology that is shown in Figure 1. It consists of one main data center and two edge clouds [22] that are at the sensor locations. The edge clouds have moderate connections to the main data center, which means additional latencies of 80 ms and a bandwidth limitation of 100 MBit/s to the main cloud and one high performance node at each site.

D. Results

For the end-to-end latencies we can observe that the search algorithms are almost twice as fast as the worst method in the comparison (see Figure 5). For the metaheuristic search methods, the steepest descent (SD) placement gives better results than the placement obtained by tabu search (TS) except for location alerts (Figure 6). The reason for this might be the optimization regarding latencies along the critical path

in the dataflow, which is the output of the fire alerts (see FireAlertMonitor at Figure 2). In addition, the path with the location reports is the one that generates the highest data load and therefore most likely causes congestions that are also optimized in the search approaches. But sometimes, the steepest descent (SD) approach might be better than the TS due to the randomly chosen placement. The SD might choose a better initial placement than the TS scheduler which leads to a better final result.

Among the constructive methods, the highest level first (HLF) approach gives better results (improvements of up to 75 ms, Figure 3), due to the structure of our benchmark job. We have high data loads emitted from the source tasks, that are filtered early to a large extend in subsequent tasks. The top down (TD) method places these tasks at the top of the network topology, so that data reduction takes place at very high levels in out network topology. This leads to a high network load on the levels before the data reduction. In contrast, the HLF approach places the tasks at the lowest possible level of the network topology, which leads to reduction of the overall data transmission.

V. Conclusion

This paper examined search-based and constructive methods for placing dataflow operators onto geo-distributed heterogeneous resources. The heuristic search-based algorithms find near-optimal placements with the help of QoS metrics regarding bandwidth, latency, and node capacities. The constructive methods determine placement assignments according to the graph structures. Our results show, that the heuristic algorithms are significantly better at obtaining schedules on topologies that are influenced by multiple factors.

Based on these findings, we want to investigate how optimal weightings of QoS terms in the optimization function can be realized in the future. Improvements can also be made in the retrieval of the QoS metrics and the requirements for dataflow tasks. Moreover, in this paper, we assumed a static environment, in which bandwidth capacities and latencies do not change over time, and also static ingestion rates, which in reality is often not the case due to for example changing network topologies and load fluctuations. Thus, continuous monitoring and adaption of schedules would considerably improve solutions for real environments and applications.

Acknowledgments

This work has been supported through grants by the German Ministry for Education and Research (BMBF) as Berlin Big Data Center BBDC (funding mark 01IS14013A and 01IS18025A).

References

[1] N. Jazdi, "Cyber Physical Systems in the Context of Industry 4.0," in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*. IEEE, May 2014, pp. 1–4.

[2] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, "An Information Framework for Creating a Smart City Through Internet of Things," *IEEE Internet of Things Journal*, vol. 1, no. 2, pp. 112–121, April 2014.

[3] M. R. Alam, M. B. I. Reaz, and M. A. M. Ali, "A Review of Smart Homes—Past, Present, and Future," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 42, no. 6, pp. 1190–1203, Nov 2012.

[4] A. V. Dastjerdi and R. Buyya, "Fog Computing: Helping the Internet of Things Realize Its Potential," *Computer*, vol. 49, no. 8, pp. 112–116, Aug 2016.

[5] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan 2017.

[6] S. Dolev, P. Florissi, E. Gudes, S. Sharma, and I. Singer, "A Survey on Geographically Distributed Big-Data Processing using MapReduce," *IEEE Transactions on Big Data (Early Access)*, pp. 1–1, July 2018.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, January 2008.

[8] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. USENIX Association, June 2010, pp. 10–10.

[9] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink: Stream and Batch Processing in a Single Engine," *IEEE Data Engineering Bulletin*, vol. 38, no. 4, pp. 28–38, July 2015.

[10] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized Streams: Fault-Tolerant Streaming Computation at Scale," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. ACM, October 2013, pp. 423–438.

[11] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle, "The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-Of-Order Data Processing," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, August 2015.

[12] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low Latency Geo-distributed Data Analytics," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. ACM, August 2015, pp. 421–434.

[13] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, "Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. USENIX Association, April 2014, pp. 275–288.

[14] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal Operator Placement for Distributed Stream Processing Applications," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, ser. DEBS '16. ACM, June 2016, pp. 69–80.

[15] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, "Storm@Twitter," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. ACM, June 2014, pp. 147–156.

[16] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-Aware Operator Placement for Stream-Processing Systems," in *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, April 2006, pp. 49–49.

[17] L. Tassiulas and A. Ephremides, "Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, Dec 1992.

[18] F. Glover, "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers and Operations Research*, vol. 13, no. 5, pp. 533–549, May 1986.

[19] H. N. Gabow, "An Almost-Linear Algorithm for Two-Processor Scheduling," *Journal of the ACM*, vol. 29, no. 3, pp. 766–780, July 1982.

[20] K. Angayarkkani and N. Radhakrishnan, "Efficient forest fire detection system: a spatial data mining and image processing based approach," *International Journal of Computer Science and Network Security*, vol. 9, no. 3, pp. 100–107, 2009.

[21] V. Vipin, "Image processing based forest fire detection," *International Journal of Emerging Technology and Advanced Engineering*, vol. 2, no. 2, pp. 87–95, 2012.

[22] L. Tong, Y. Li, and W. Gao, "A Hierarchical Edge Cloud Architecture for Mobile Computing," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. IEEE, April 2016, pp. 1–9.