# Governor: Operator Placement for a Unified Fog-Cloud Environment

Ankit Chaudhary
TU Berlin
ankit.chaudhary@tu-berlin.de

Steffen Zeuch
DFKI GmbH, TU Berlin
steffen.zeuch@dfki.de

Volker Markl
DFKI GmbH, TU Berlin
volker.markl@tu-berlin.de

## ABSTRACT

The processing of geo-distributed data streams is a key challenge for many Internet of Things (IoT) applications. Cloud-based SPEs process data centrally and thus require all data to be present in the cloud before processing. However, this centralized approach becomes a bottleneck for processing data from millions of geo-distributed sensors on a large scale IoT infrastructure. A new line of research extends the centralized cloud with decentralized fog devices to mitigate this bottleneck. One major challenge for an SPE in this unified fog-cloud environment is to fulfill user requirements by placing operators on fog or cloud nodes.

In this demonstration, we introduce *Governor*, an operator placement approach for a unified fog-cloud environment. Our approach consists of the *Governor placement process* and *Governor policies* (GPs). The Governor placement process utilizes heuristic-based GPs to optimize operator placement for a user query. Using GPs, administrators can control the operator placement process to fulfill specific Service-Level-Agreement (SLA). We implement Governor in the NebulaStream Platform (NES), a data and application management system for the IoT. We showcase the impact of GPs on operator placement for different example queries. Our demonstration invites participants to simulate the operator placement of queries and discover their characteristics. This demonstration represents a first step towards an efficient operator placement approach for upcoming IoT infrastructures with millions of sensors and thousands of queries.

## 1 INTRODUCTION

Over the last decade, the adoption of IoT devices has increased significantly [7]. Processing IoT data in real-time enables a wide range of new opportunities for businesses (e.g., smart homes, connected cars, health-care) [13]. Many IoT applications today are implemented using a cloud-based infrastructure. To perform the data processing, a continuous transfer of geo-distributed IoT data to a centralized data-center is required. Data processing frameworks such as Flink [2] and Spark [12] are designed for cloud-based environments and support efficient data analytics and virtually unlimited scaling. However, cloud-based processing of geo-distributed IoT data presents challenges for real-time and latency-sensitive applications. These challenges include high data transmission costs, delayed data processing, and high demand for cloud-resources [13].

Fog computing addresses these shortcomings by processing data closer to the source devices [1]. In particular, fog computing leverages intermediate compute nodes to perform data processing and thus minimizes data transfer between source IoT devices and cloud servers. However, in contrast to the robust and elastic cloud resources, the fog consists of limited, unreliable, and low-end heterogeneous nodes. On the one hand, the fog can apply in-network processing to reduce large volumes of data. On the other hand, the fog has to cope with unreliable nodes and intermittent failures [11, 13].

The NebulaStream Platform provides data analytics capabilities in a unified fog-cloud environment [13]. The proximity of computing resources to IoT devices in the fog, combined with nearly unlimited computing resources in the cloud, presents novel opportunities for IoT data processing and holistic optimizations. One major challenge in such an environment is the placement of query operators on compute nodes with regards to the unique infrastructure characteristics and specific SLA requirements. Recent work on operator placement in the cloud focuses mainly on network and compute resource efficiency but does not take the volatility and heterogeneity of the fog infrastructure into account [5, 8, 10]. In contrast, approaches for unified fog-cloud environments consider volatility and heterogeneity but only optimize for a specific goal, e.g., network efficiency or fault-tolerance [3, 4, 6]. Furthermore, current approaches do not allow administrators to specify SLA objectives (e.g., high-throughput, low resource consumption) for operator placement.

In this demonstration, we introduce *Governor*, a new fog-cloud operator placement approach that allows specifying custom SLA objectives. Our approach consists of a set of heuristic-based rules called *Governor policies* and a two-phase *Governor placement process*. Using GPs, we enable administrators to tune the Governor placement process for specific SLAs. The two phases of the Governor placement process are the following. First, the *path selection* phase identifies a set of paths between sensors, intermediate compute nodes, and the cloud. Second, the *operator assignment* phase assigns operators to the compute nodes residing on the identified paths. In this demonstration, we showcase our placement approach and the impact of different GPs using five example scenarios. Attendees of our demonstration can compare operator placements of custom queries for different GPs. Overall, our approach allows administrators to guide operator placement using GPs and finds effective operator mappings for large query plans over millions of sensors. In summary, our contributions are as follows:

(1) We introduce Governor for performing operator placement in a unified fog-cloud environment.
(2) We present five Governor Policies for example scenarios.
(3) We present a user interface that allows attendees to study the impact of different GPs on operator placement.

The rest of this paper is structured as follows. In Section 2, we introduce our Governor approach. In Section 3, we present our demonstration scenario and describe the overall system design. We discuss related work in Section 4 and conclude in Section 5.

## 2 GOVERNOR

Governor consists of Governor Policies and the Governor placement process. We introduce GPs in Section 2.1 and the Governor placement process in Section 2.2. After that, we showcase the

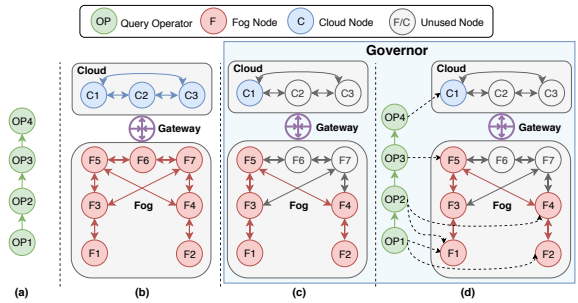


**Figure 1: Governor placement process.**

application of the Governor approach using example scenarios (see Section 2.3).

## 2.1 Governor Polices

*Governor Policies* are a fixed set of heuristic-based rules that guide the operator placement of a query. For example, to place a query with a SLA-objective high fault-tolerance SLA-objective, a GP selects all available network paths between source and sink nodes, and places replicated operators on different nodes along the selected paths. In case of an operator failure, another replica operator will take over the failed operator's workload, thereby achieving the fault-tolerance objective. Using GPs, the operator placement process will optimize the query execution for a specific SLA-objective. An administrator prepares the GP by selecting rules from a predefined catalog, which is maintained and updated by a domain expert. The rules are classified into two categories: *path selection* and *operator assignment*. The path selection rules guide the selection of a subset of all available paths. In contrast, the operator assignment rules guide the placement of operators on nodes on the selected paths. In general, a GP contains at least one rule from each category. However, an administrator can define several GPs, each aiming to optimize query placement for a unique SLA-objective using its distinct set of rules. Additionally, while different GPs can share individual rules, the combination of rules within a GP is unique.

## 2.2 Governor Placement Process

In this section, we describe how the Governor placement process performs the operator assignments while taking GPs into account. Figure 1 shows the steps necessary to place the operators of a query on a physical infrastructure. First, NES transforms a user query into a directed acyclic graph (DAG) that is composed of query operators and directed links among them, as shown in Figure 1(a). The query DAG Q is represented by $Q = \{O, L\}$ where O and L represent operators and directed links respectively. In the background, NES maintains an infrastructure graph containing compute nodes that are interconnected by network links (see Figure 1(b)). The infrastructure graph G is represented as $G = \{V, E\}$ where V and E represent vertices and edges respectively. Second, NES provides the query DAG, the infrastructure graph, and a GP to Governor for operator placement. Governor uses its placement process to assign the DAG operators on the infrastructure nodes while following the rules from GP.

The operator placement process consists of two phases: the *path selection phase* and the *operator assignment phase*. In the path selection phase, shown in Figure 1(c), Governor identifies the path between sources (IoT device) and the sink (cloud servers) node using the heuristics defined in the GP. The number of paths that are selected in the path selection phase ranges from all paths to just a specific path, and thus reduces the search space for the operator assignment phase. The path selection phase is crucial for

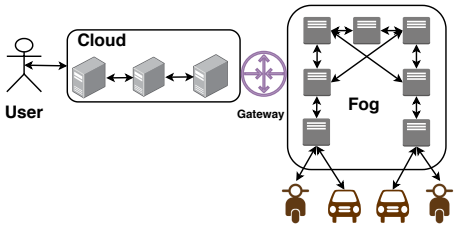


**Figure 2: Infrastructure for a ride sharing services.**

large-scale IoT infrastructures containing thousands of heterogeneous nodes and millions of data sources. We make use of depth first search (DFS) algorithm for finding paths. The maximum number of source nodes in G is given by total number of vertices $|V|$. The worst-case runtime complexity for identifying paths between $|V|$ sources and sink node is represented by $O(|V|^3)$.

In the operator assignment phase, shown in Figure 1(d), Governor performs the operator placement on nodes along the selected paths. During the operator assignment phase, Governor distinguishes between *pinned* and *unpinned* operators. Pinned operators reside on a specific location, e.g., source operators on IoT devices and sink operators on cloud servers. In contrast, unpinned operators can be placed on any node along the selected paths (if resource constraints permit this). Common strategies would place non-blocking operators (e.g., filter, source, sink) close to the IoT devices and blocking operators with state (e.g., window, aggregation, join) close to the cloud servers. For a query DAG Q with $|O|$ operators and $|P|$ selected paths with average $|N|$ nodes, the worst-case runtime complexity of operator assignment phase is given by $O(|O| * |N| * |P|)$.

## 2.3 Governor in Action

In Figure 2, we show a representative IoT infrastructure for ride-sharing services such as ShareNow[1], WeShare[2], or Coup[3]. Vehicles with on-board computing units communicate with fog computing devices and transmit the operational information (e.g., location, user, or time information) to the fog infrastructure. While data is flowing through the fog into the cloud, fog nodes can apply processing. Users interact with the cloud-based system using a mobile application for locating, renting, or accessing various other services. Using this infrastructure, we present five example application scenarios:

(1) **Fast-response:** Realtime tracking of vehicle fleet.
(2) **Fail-safety:** Billing at the end of a trip.
(3) **Bursty-data:** Monitor vehicle statistics during usage.
(4) **Save-resource:** Health checks on IoT infrastructure.
(5) **Save-energy:** Save energy on battery-operated vehicles.

The presented scenarios have different SLA requirements, which guide their respective placement of operators. In Table 1, we present five example GPs for these scenarios. The *Low-Latency*, *Fault-Tolerance*, and *High-Throughput* GPs are focused primarily on the performance of queries. In contrast, the *Minimum Resource Consumption* and *Minimum Energy Consumption* GPs are focused on the performance of the infrastructure nodes. In the following, we use GPs from Table 1 and briefly discuss the placement process for the five application scenarios.

**Fast-response** requires fast event processing and delivery by performing early data computation and using low latency network links. The *Low-Latency* GP from Table 1 satisfies this SLA requirement. In the path selection phase, Governor selects distinct paths with low link latencies between source and sink

[1] www.sharenow.com [2] www.weshare.com [3] www.coup.com

| | Low-Latency | Fault-Tolerance | High-Throughput | Minimum Resource Consumption | Minimum Energy Consumption |
|---|---|---|---|---|---|
| **Path Selection Phase** | • Distinct paths with low link latency | • All paths between sources and sink | • Distinct paths with high bandwidth capacity | • Common path between sources and sink | • Common path between sources and sink |
| **Operator Assignment Phase** | • Non-blocking operators closer to source<br>• Replicate operators when possible | • Use shared nodes among selected paths<br>• Replicate operators when possible | • Non-blocking operators closer to source<br>• Blocking operators closer to sink | • Share intermediate operators among different sources<br>• Avoid operator replication | • Non-blocking operators closer to source<br>• Share intermediate operators among different sources<br>• Avoid operator replication |

**Table 1: Five example Governor policies.**

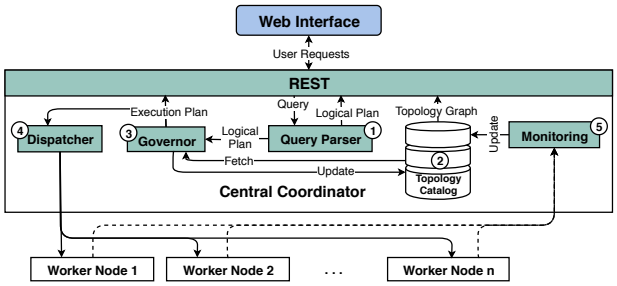


**Figure 3: Demonstration system architecture.**

nodes. In the operator assignment phase, Governor places all non-blocking operators close to the source operators and replicates the operators wherever possible to achieve high data parallelism [9].

**Fail-safety** requires events to be delivered irrespective of intermittent network or node failures by using alternative nodes or links for processing and data transfer. The *Fault-Tolerance* GP from Table 1 satisfies this SLA. In the path selection phase, Governor selects all possible paths between the sources and the sink nodes. In the operator assignment phase, Governor places operators on the nodes shared across multiple paths such that in the event of a path failure, another network path can be used for data delivery.

**Bursty-data** requires the query to handle a sudden burst of events by transmitting data using a high bandwidth link. The *High-Throughput* GP from Table 1 satisfies this SLA. In the path selection phase, Governor selects distinct paths with high bandwidth capacity between source and sink nodes. In the operator assignment phase, Governor places all non-blocking operators close to the source operator and all blocking operators close to the sink operator.

**Save-resource** requires the compute node to save resources by sharing query operators and preventing replications. The *Minimum Resource Consumption* GP from Table 1 satisfies this SLA. In the path selection phase, Governor selects a common path between source and sink nodes. In the operator assignment phase, Governor ensures a high degree of operator sharing among multiple sources contributing to the query.

**Save-energy** requires the compute nodes to save power either by reusing the existing query operators or by reducing the amount of data transmitted over the network. The *Minimum Energy Consumption* GP from Table 1 satisfies this SLA. Like for **Save-resource**, Governor selects a common path between source and sink nodes. However, in the operator assignment phase, Governor tries to place non-blocking operators closer to the source to reduce downstream data traffic and thus saving energy for transmission.

Governor uses a greedy approach for the operator placement, which can return a solution in a reasonable amount of time at the cost of sub-optimal placement decisions. In the future work, we will examine the response time-optimality trade-off in detail. Although Governor uses one query plan as an input, this query plan can be the outcome of a multi-query optimization process and thus may contain operators from multiple queries.

# 3 DEMONSTRATION

In Section 3.1, we introduce the architecture of our demonstration. After that, we present the web interface and describe the functions available for the attendees to explore in Section 3.2.

## 3.1 Demonstration System Architecture

In Figure 3, we present the interaction among the web interface, the central coordinator, and the worker nodes. The web interface submits queries and retrieves information from NES over a REST interface, e.g., the DAG for currently deployed query or the infrastructure topology. The central coordinator node manages both the query placement and the deployment process. The worker nodes manage the execution of operators from different queries. We refer the reader to relevant literature for a detailed design overview of NES [13]. In this section, we will only cover aspects that affect the operator placement process of NES.

Inside the coordinator node, the queries are first validated and parsed into DAGs using the *Query Parser* ①. After that, Governor receives the logical query plan and fetches the information from the *Topology Catalog* ②. The topology catalog maintains the latest infrastructure graph, including information on resource availability and the set of deployed query operators on individual infrastructure nodes. Based on this information, *Governor* ③ creates the execution plan, which contains the operator placement information. Finally, the *Dispatcher* ④ receives the overall execution plan and forwards it to the respective worker nodes.

In an asynchronous second feedback loop, Governor updates the topology catalog with the operator placement information. Furthermore, worker nodes send regular updates to the *Monitoring* ⑤ system, which in turn updates the topology catalog.

## 3.2 Web Interface

Figure 4 shows the web interface that attendees can use to explore operator placement with Governor. Attendees have to write their queries into the text panel ① to explore different functionalities of the web interface. The central coordinator system from Figure 3 is responsible for parsing, validating, and returning the DAG for the input query. The button-panel ② on the web interface presents various options for interacting with the components of the demonstration system. First, attendees can click on the *Show Topology* button to fetch the infrastructure graph representing the latest state of compute nodes and network connections among them. The display panel ④ shows the returned infrastructure graph. Second, the button *Show Query Plan* triggers the query parser component, which returns a DAG of interconnected operators for the submitted user query in the text panel ①. The display panel ③ shows the query DAG. Third, the drop-down button *Execution Plan* presents a set of GPs for triggering Governor. Attendees can choose between any of the available GPs shown

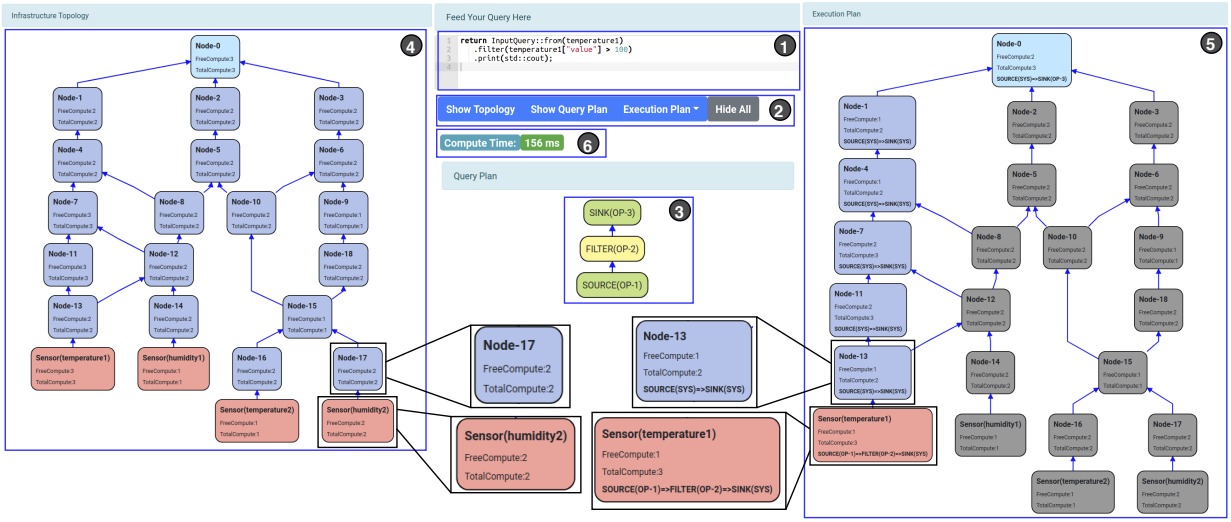


**Figure 4: Demonstration system web interface.**

in Table 1, i.e., Low-Latency, Fault-Tolerance, High-Throughput, Low Energy Consumption, and Low Resource Consumption. The display panel ⑤ will show the operator placement plan returned by Governor. The operator placement plan contains pinned, unpinned, as well as system-generated operators (e.g., forward operators) and their corresponding mappings on infrastructure nodes. Additionally, the time taken for the plan computation using the selected GP is shown in the UI ⑥.

Overall, attendees of our demonstration can explore the operator placement in a unified fog-cloud infrastructure. We will point out different options and their impact on particular characteristics. Using this demonstration, we present a first step towards an operator placement designed for upcoming IoT infrastructures with millions of sensors and thousands of queries.

## 4 RELATED WORK

In this section, we categorize related work from different research areas. One line of research covers operator placement approaches for a centralized computing infrastructure. Huang et al. offer a heuristic-based operator placement approach for optimizing with latency and throughput [8]. Kafil et al. consider heterogeneity within a distributed environment to find one optimum compute node to place all operators together [10]. Chatzistergiou et al. optimize the operator placement for inter-node network bottlenecks by reducing the number of nodes required to running a query [5]. In contrast, Governor allows a flexible and controlled operator placement strategy by enabling administrators to specify different optimization objectives within the same infrastructure.

Another line of research examines placement approaches for a unified fog-cloud environment. The approach presented by Veith et al. focuses mainly on minimizing total data processing latency [6]. However, this approach does not consider the node and link volatility within the fog as a factor that can significantly impact query execution. Cardellini et al. present a multi-objective operator placement approach [4]. However, their approach is not optimized for computing operator placements for a large scale IoT infrastructure. In contrast, Governor considers various characteristics of the fog-cloud infrastructure, including the high volatility that is commonly present in a fog infrastructure. Furthermore, Governor's two-phase approach using heuristics reduces the computation time for operator placement.

Overall, none of the previous approaches support defining flexible operator placement objectives. In contrast, Governor allows the administrator to specify different optimization goals using GPs for each query submission.

## 5 CONCLUSION

In this demonstration, we present Governor, a first step towards operator placement on IoT infrastructures with millions of sensors and thousands of queries. We demonstrate Governor's ability to accept custom Governor Policies with different optimization objectives for operator placement. In addition, we present five example policies and showcase their placement for five example application scenarios. In our demonstration, we will present the challenges as well as early solutions for operator placement in the future IoT era.

## 6 ACKNOWLEDGEMENT

This work was funded by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref 01IS18037A).

## REFERENCES

[1] Bonomi et al. 2012. Fog computing and its role in the internet of things. In *MCC*. ACM.
[2] Carbone et al. 2015. Apache flink: Stream and batch processing in a single engine. *IEEE Computer Society TCDE* 36, 4.
[3] Cardellini et al. 2016. Optimal operator placement for distributed stream processing applications. In *DEBS*. ACM.
[4] Cardellini et al. 2017. Optimal operator replication and placement for distributed stream processing systems. *SIGMETRICS* 44, 4 (2017).
[5] Chatzistergiou et al. 2014. Fast heuristics for near-optimal task allocation in data stream processing over clusters. In *CIKM*. ACM.
[6] da Silva Veith et al. 2018. Latency-aware placement of data stream analytics on edge computing. In *ICSOC*. Springer.
[7] de Assuncao et al. 2018. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications* 103 (2018).
[8] Huang et al. 2011. Operator placement with QoS constraints for distributed stream processing. In *CNSM*. IEEE.
[9] Jaspal et al. 1993. Exploiting task and data parallelism on a multicomputer. In *ACM SIGPLAN Notices*, Vol. 28. ACM.
[10] Kafil et al. 1998. Optimal task assignment in heterogeneous distributed computing systems. *IEEE concurrency* 6, 3 (1998).
[11] O'Keeffe et al. 2018. Frontier: Resilient edge processing for the internet of things. In *VLDB*.
[12] Zaharia et al. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*. USENIX Association.
[13] Zeuch et al. 2020. The NebulaStream Platform: Data and Application Management for the Internet of Things. In *CIDR*.