Shepard: Dynamic Placement of Microservices in the Edge-Cloud Continuum

Farhan Asghar, Tehreem Fatima, Junaid Haroon Siddiqui,
Naveed Anwar Bhatti(✉), and Muhammad Hamad Alizai

Lahore University of Management and Sciences LUMS, Lahore, Pakistan
{18030017,18030009,junaid.siddiqui,naveed.bhatti,hamad.alizai}@lums.edu.pk

Abstract. We present Shepard, an innovative microservice placement approach tailored for edge-assisted cloud infrastructures. Shepard dynamically migrates application services between the edge and cloud to harness optimal performance gains. This approach is structured around three core components: (1) a *resource manager* for monitoring available edge resources and the evolving demands of applications, (2) an *optimization module* that transposes the service placement dilemma into a labeled-graph cut challenge, aiming to identify the most advantageous cut given a set of parameters, and (3) a *deployment module* tasked with adjusting service placement in response to shifts in the optimal graph cut's position.

Our implementation of Shepard underwent rigorous testing in two distinct case studies. In the inaugural study, Shepard managed energy for a solar-driven edge within an agricultural IoT framework, resulting in a striking 79% elevation in service reliability and availability compared to a conventional static service placement strategy. For our subsequent study focusing on cost-effectiveness within a ride-hailing application, Shepard facilitated a substantial 45% slash in application deployment expenses, all the while maintaining comparable performance levels to a standard dynamic service placement technique.

Keywords: Edge Computing · Microservices-based Architecture · Dynamic Resource Management

1 Introduction

The evolution of edge computing has created the need for new ways of structuring and deploying application services. An edge is not just a homogeneous extension of cloud resources that can scale up application services on demand, but a precious compute resource near the data source. Intelligently managing this limited yet expensive compute resource is imperative to enhance application performance or extend the advantages of edge to multiple applications (or tenants). As such, deciding what part of the application logic goes into the edge and what stays in the cloud has become a key DevOps challenge.

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2024
Published by Springer Nature Switzerland AG 2024. All Rights Reserved
A. Zaslavsky et al. (Eds.): MobiQuitous 2023, LNICST 594, pp. 43–62, 2024.
https://doi.org/10.1007/978-3-031-63992-0_4

A static decomposition and placement of services is not feasible because the factors influencing this decision are dynamic, such as the current load of the service, the available resources on the edge and their cost, and the service's performance requirements [43]. Although solutions exist that dynamically scale up or scale down services across data centers or cloud platforms, they are not cognizant of the specific features of edge-assisted clouds. Therefore, developing a practical solution to the service placement problem in this domain holds a much greater significance to garner the unique benefits of edge computing.

Challenges. An edge corresponds to the compute, storage, and network resources available in the request's geographical proximity. Augmenting a cloud with an edge resource is becoming essential to meet the growing latency, availability, bandwidth, and privacy requirements of modern applications, such as artificial intelligence models, augmented and virtual reality, the internet of things, robotics, and video analytics. However, service orchestration and placement strategies in an edge-assisted cloud infrastructure need to address challenges that go well beyond the typical scaling of services within a cloud for the following reasons.

- *Finite Resources:* Resources on the edge cannot anticipate unlimited capacities, and they may vary depending on the characteristics of the edge platform, which could be an extremely resource-constrained smart object, a cellular gateway, or a reasonably well-provisioned small-scale data center (cloudlet). Thus, resources on the edge must be carefully managed and intelligently utilized.
- *Dynamic Requirements:* Application requirements may change over time and space due to fluctuating loads and mobility of requests. A service with sharp latency requirements at one time of the day may become mostly irrelevant during other times when another service comes to the fore [24]. Therefore, service provisioning on the edge must be amenable to such dynamic application requirements.
- *Differential Costs:* Unlike clouds, the cost of the edge may vary based on the current load [23] or ownership. The edge may be owned by the same or different cloud service providers, which may employ surge pricing to achieve supply-demand equilibrium [39], or it may be wholly free but severely resource-constrained if owned by the end-user. Thus, a cost-aware strategy is needed for affordable service orchestration in the edge-assisted cloud.

Scope. The key to addressing this challenge is to determine *when* and *which* portion of the application logic should reside on the edge. Scaling or migrating one service to the edge may require bringing another one back to the cloud to create space for the former or reduce total cost. This may entail continuous monitoring of edge resources and assessing application requirements over time. For instance, in the context of IoT, an occupant of a smart home uses different appliances at different times of the day [31], providing room for dynamically reconfiguring related services on the edge for real-time response. Changing vehicular traffic conditions in a city may provide an opportunity for smart city applications to

migrate the adaptive signal-control service [40] back from the edge to the cloud to reduce deployment costs. A solar-powered edge may accept different loads depending on the predicted energy provisioning patterns [44].

Solution. We present `Shepard`, a microservices-based service orchestration and placement architecture for edge-assisted cloud infrastructure. The use of microservices provides a fundamental underpinning for decomposing applications into loosely coupled, independent services. The `Shepard` architecture comprises three components: First, a *resource manager* that (i) monitors available resources on the edge and (ii) gathers statistics about each service's demands and performance. Second, the *optimization module* that maps the services and their interconnections into a graph, thus transforming the service placement problem into a graph cut problem to find the best cut (placement decision for a set of services) given a set of constraints (resource availability on the edge). This module populates the vertices and edges of the graph with information accrued by the resource manager. Third, a *deployment module* that migrates services between the edge and the cloud and reconfigures them whenever the location of the best cut in the graph changes.

Thus, we make the following concrete contributions.

- We propose and implement a novel end-to-end service orchestration architecture for edge-assisted cloud infrastructure.
- We extensively evaluate this architecture in two case-studies and compare its performance with a baseline dynamic placement approach.

Benefits. Our first case-study demonstrates how `Shepard` improves the availability of services on a solar-powered edge deployed at a remote agricultural farm. We observed up to a 79% decrease in application downtime compared to the original static placement of services, resulting in fewer node deaths, reduced application latency, and increased availability and reliability of services. Our second case-study shows how `Shepard` reduces application deployment costs on the edge by dynamically migrating unused or overloaded services to the cloud while delivering almost edge-native performance to the end-user. Our results show a 45% reduction in application deployment costs in an edge-assisted cloud infrastructure while ensuring cheaper computation and lower latency than a baseline dynamic service placement strategy.

2 Background and Motivation

Edge Computing. The importance of edge computing in modern cloud infrastructures is undisputed for a wide range of applications. In edge-assisted clouds, the edge serves as the first computing point for data. This allows computations to be performed closer to the source of data, resulting in immediate response times, better learning, and caching, rather than transferring the data to the cloud. Our definition of edge is not limited to a specific platform, as in existing literature [9], but includes any computing unit near the last mile network where

a subset of application services can be offloaded. An edge may be an IoT device, a gateway to the internet, or a slim datacenter nearby.

As a result, service placement challenges may vary depending on the type and capabilities of the edge platform.

Service Placement. Service placement is the process of mapping services (or virtual machines) to available physical hosts within a datacenter or a remote datacenter in a cloud federation. Various efficiency goals dictate these placement strategies. For example, load balancing [11,38] can improve application performance, while minimizing the number of physical machines needed to run the application(s) and shutting down unused ones can enhance energy efficiency [1,3,16].

In contrast to datacenter-based clouds, edge computing does not have the luxury of infinite resources [18,25,45]. The edge may have limited energy, computation, memory, storage, and networking capabilities, as it is designed to serve a subset of users in close geographic proximity. As a result, service placement becomes a challenge. The edge may only be able to host a subset of application services, which must be identified based on a well-defined set of metrics that ensure performance gains. For instance, a low-latency service may be better placed in the cloud if the compute delays on the edge exceed the network delays to the cloud [33]. Additionally, the cost of these services and the availability of resources on the edge may change over time due to varying workloads [32]. There may also be further restrictions on the mobility of services, such as a privacy-preserving service that cannot be moved to the cloud [28].

A service placement strategy in edge-assisted clouds must consider these unique challenges.

Microservices. Microservices-based cloud application designs are a growing trend nowadays [7,27]. Most cloud service providers have successfully transitioned from monolithic architecture to microservices [2,8,26]. This is primarily because microservices resolve traditional limitations of monolithic designs in handling huge codebases, adopting new technology, scaling up deployments, implementing more recent functionality, and more.

Microservices enable applications to be developed as a suite of small services that run in their own processes and communicate using lightweight mechanisms. They are typically packaged in a container that encapsulates the code and all its dependencies, allowing for reliable migration of a service from one computing environment to another and quick resumption.

The introduction of edge computing further emphasizes the need for microservices. Monolithic architectures do not meet the increased flexibility requirements in scaling, profiling, placement, and reconfiguration of application modules. The idea of edge-assisted cloud infrastructure seamlessly integrates with microservices-based application design. The latter enables applications to behave as a single entity, yet have several dissociated parts. These containerized parts can then perform their designated functions irrespective of their location in the system, be it inside the cloud datacenter or at a resource-constrained edge in a different continent.

Consequently, our proposed solution considers microservice-based application designs to keep up with existing trends and unveil the great potential of edge computing.

3 Shepard

We first provide a brief overview of the design space and the associated challenges involved in developing a dynamic service placement solution. We then describe, in detail, the design of each architectural component of Shepard. Finally, we conclude this section by providing the most relevant implementation details.

3.1 Overview and Challenges

Dynamic service placement techniques are commonly employed in data-center based clouds. However, existing approaches are not tailored to the specific needs of an edge-assisted cloud architecture, where an edge is not necessarily a homogeneous extension of existing cloud resources. To achieve the desired performance benefits and the vision of edge computing, the resources on the edge must be intelligently utilized.

Resource Monitoring. In this context, a service placement strategy must account for the dynamic demands of heterogeneous services constrained by resources' availability on the edge. It should identify the most suitable services in the cost-benefit spectrum at any given time and accordingly reconfigure their placement on the fly. To achieve this, we first need to identify candidate services that can be moved from the edge to the cloud and vice versa.

Shepard collects different metrics that define the performance of a service in a given application. These metrics are collected at predefined intervals and recorded into a data store, where they can be queried to assess performance and potentially migrate services if the placement strategy determines a more profitable configuration than the current one. For this purpose, each host runs a lightweight monitoring service that gathers statistics such as the number of calls across pairs of services, the amount of data transferred, CPU usage, average memory consumption, etc. Shepard periodically analyzes service performance and resource utilization on the edge to make placement decisions.

Placement Strategy. A greedy approach to service placement would either place all the services on the cloud or edge, for example, to reduce network bandwidth requirements. Similarly, a load-balancing approach to service placement may proportionally distribute services across the edge and cloud. However, this is not the goal in an edge-assisted cloud infrastructure. We need to find a placement layout that maximizes the desired performance metrics of a given application.

Shepard maps the services onto a labeled graph such that the vertices of the graph represent services and the edges represent the data transfer between different services. The labels on vertices capture the resource demands of a service, while the labels on edges capture the bandwidth requirements, including

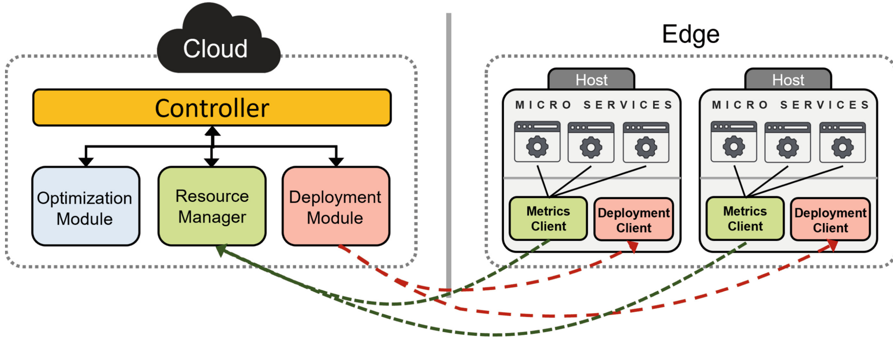


Fig. 1. System Architecture

the number of calls and the amount of data transferred between pairs of services. These labels are periodically updated.

If the application dictates certain latency requirements while minimizing bandwidth requirements, `Shepard` can partition (or cut) the graph in a way that achieves this optimization goal. The latency-critical services will be grouped in a single partition and dispatched to the edge. The best partition (min-cut) in the graph, however, may change over time as the labels are updated. Moreover, the partitioning process must also take into account the resource constraints of edge nodes.

A brute force technique that exhausts the entire search space is a trivial solution to finding the best cut in a graph, but its complexity increases exponentially with the size of the graph. Appropriate heuristics must be employed by `Shepard` to reduce this complexity without compromising the accuracy of finding the best cut. To this end, `Shepard` reduces the size of the graph by aggregating edges, for example, by combining vertices with the highest weights between their edges. The effect of such aggregation naturally places two related services either on the edge or in the cloud while reducing the search space size for finding the best cut in linear time.

Deployment. After the graph has been partitioned, `Shepard` deploys the services according to the determined placement strategy. This may involve scaling up services due to increased demand or moving services between the cloud and edge. There are different migration strategies that can be employed, such as moving services entirely, but this may result in significant network bandwidth consumption. Alternatively, services can be stored locally and activated by `Shepard` to minimize migration overhead, although this may not be feasible for stateful services. `Shepard` uses a combination of these methods to maximize their benefits whenever possible.

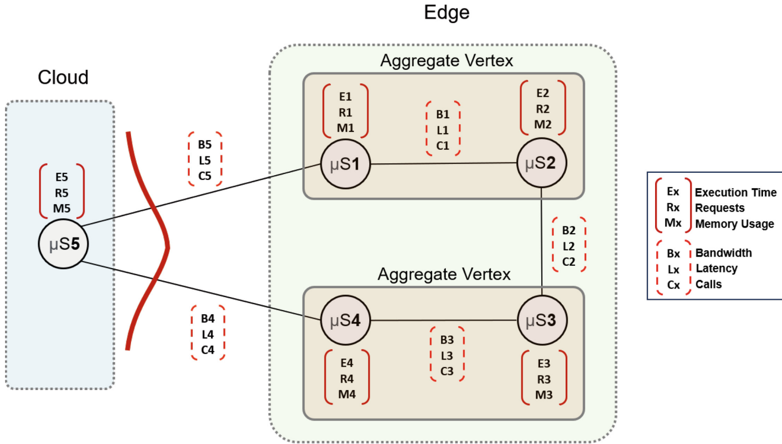


Fig. 2. Capacity-aware multi-weighted min-cut: Visualization of vertex aggregation based on high-weight edges to optimize service placement.

3.2 Architecture

The high-level architecture of `Shepard` is illustrated in Fig. 1. `Shepard` consists of three modules responsible for monitoring resource requirements, optimizing placement decisions, and deploying microservices. These modules are integrated through the `Shepard` controller, which configures them on an application-specific basis and facilitates information exchange among them.

Resource Manager. The Resource Manager module maintains a record of the available resources on the edge and the dynamic resource demands of microservices. It relies on a resource manager client installed on each edge to gather these statistics and relay them back to the main module in the cloud. The metrics of interest for this module include available resources on the edge, such as computing power, memory, storage, and network bandwidth, as well as application performance metrics, such as execution time, network latency, and the number of service calls and data exchanged between service pairs. This information is essential for the optimization module to generate a comprehensive view of the current service deployment configuration and suggest changes to accomplish specific optimization objectives, such as reducing deployment cost or latency.

Optimization Module. To devise an efficient placement plan, the optimization module first constructs a labeled graph using the resource manager's metrics through the following approach.

Graph Construction: Each microservice is depicted as a vertex in the graph, while the edges signify the coupling between services. In Fig. 2, we provide a representative example with six vertices: two on the cloud and four on the edge. The vertices in this figure are marked with the resource demands of the corresponding service, which include memory consumption, execution time, and the

average number of requests to be served. The labels on the edges indicate the communication cost between services. This representation provides a comprehensive view of the application requirements and the extent to which they can be fulfilled on resource-constrained edges. While our illustration centers on some primary constraints, it's worth noting that specific application scenarios might introduce additional requirements. For instance, in settings where edge devices operate on harvested energy, battery levels become a paramount constraint. Similarly, when the cost associated with edge usage fluctuates over time, temporal factors emerge as crucial constraints. The optimization module then finds the optimal cut in the graph, while adhering to the edge constraints, and returns the set of services to be deployed on the edge. The remaining services are deployed on the cloud. A deeper dive into these variable constraints and their implications will be elaborated upon in the evaluation Sect. 4.

Brute Force: A brute force technique for finding the optimal service placement layout involves exhaustively searching the solution space. This method generates all possible partitions in the graph and selects the one that allocates the maximum number of services to the edge while satisfying the resource and cost constraints. However, this straightforward approach does not scale well because it requires exploring a potentially large solution space before identifying the optimal partition.

Algorithm 1. contractEdge

Require: $G = (V, E)$
Ensure: Graph G'
  while Number of nodes in graph G' > THRESHOLD **do**
    $e \leftarrow$ edge from G with the highest weight
    $\{v1, v2\} \leftarrow$ the two vertices associated with edge e
    create a new vertex v
    assign v a weight equal to the sum of weights of $v1$ and $v2$
    $G' \leftarrow G$ with e removed
    Remove $v1$ and $v2$ from G'
    Add vertex v to G'
    for each edge in G' connected to $v1$ or $v2$ **do**
      Redirect the edge to v
    end for
  end while
  return $G' = 0$

Heuristic Algorithm: In light of the scalability challenge posed by the brute force method, we introduce a heuristic approach. This heuristic is rooted in the observation that pairs of services, which frequently communicate or have a high data exchange rate, placing them close (either both on the edge or both on the cloud) can be optimal for performance. Thus, our heuristic aggregates those vertex pairs in the graph with exceedingly weighty edges, considering these unlikely to form a cut. Figure 2 not only demonstrates the graph construction but

also showcases the vertex aggregation of the heuristic method. In this figure, the four vertices on the edge are consolidated into two, highlighting the compactness achieved by our heuristic approach.

Algorithm 1 details the process of aggregating vertices. It iteratively contracts edges with high weights and combines their vertices into a single virtual node until a pre-defined size of the graph is achieved. The compact, aggregated graph is then used as input to the brute force search, which is described in Algorithm 2.

To reinforce the effectiveness of our heuristic, Fig. 3 offers a performance comparison between the brute force method and the heuristic-based by showing the search times for the optimal min-cut value of both algorithms. This figure specifically illustrates the execution times of both algorithms with varying numbers of microservices. For the initial testing and performance evaluations illustrated in Fig. 3, both algorithms made use of synthetic microservices. These synthetic services were designed to replicate typical microservice scenarios and the varying parameters as previously discussed.

It is important to note that the brute force search time rises exponentially with the number of microservices, while the heuristic method exhibits a more favorable linear growth. Remarkably, despite the differences in computational complexity and time, both the brute force and heuristic algorithms converged to the same min-cut value, affirming the heuristic's efficacy.

Algorithm 2. Find Min-Cut

Input: A weighted graph $G = (V, E)$ with node weights W and edge constraints (EC)
Output: Set of services to be deployed on edge (S)
    Initialization:
1: S = {}
    Procedure:
2: G' = contractEdge(G, e)
3: Find the weights W of all possible cuts $C = (A, A')$ in G' based on edge weights E' where $A'$is deployed on edge
4: Sort the cuts C in ascending order of weights W
5: **for** *cut* in C **do**
6:     **if** (Sum of Vertex weights of services $A' \leq$ Edge Node Capability (EC)) **then**
7:         $S = S \cup A'$
8:     **end if**
9: **end for**
10: **return** S =0

Deployment Module. The deployment module merely updates the service placement layout based on the output of the optimization module. Depending on the underlying platform, this may require migration of services between cloud and edge or only instigating a service stored locally. Further constraints may also apply, such as migration of the state of a state-full service even if the code is stored locally.

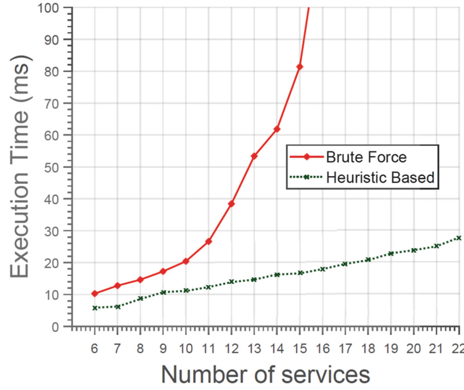


Fig. 3. Performance comparison of a brute force strategy with a heuristic based approach

3.3 Implementation

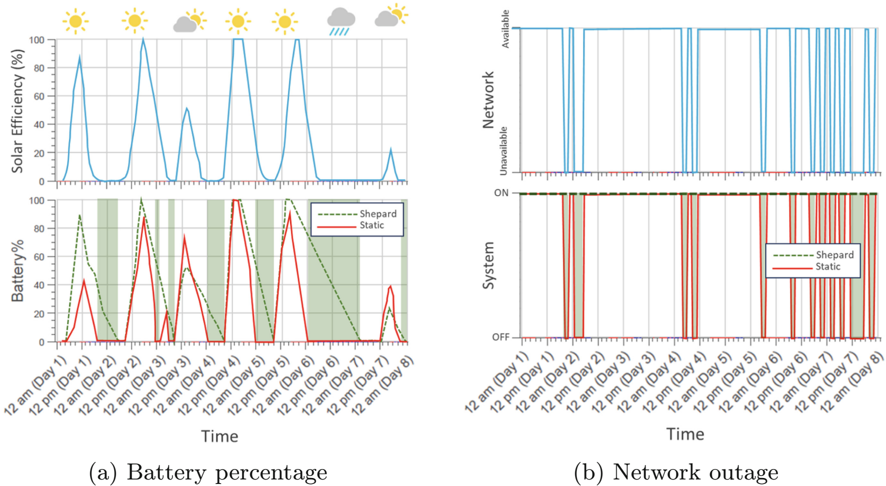
We implemented `Shepard` using Nodejs [10] and utilized Docker containers as microservices, given their popularity in both industry and academia. A container is a standardized software unit that packages code and its dependencies, including the runtime, system tools, system libraries, and settings, to ensure reliable operation on diverse computing environments [5]. Different services communicate with each other using sockets [36], and these Docker containers run on a Docker engine that provides a secure and scalable isolated environment [5,14]. Our implementation assumes that each microservice is running in a separate Docker container [5].

We implemented our *deployment module* using Docker Swarm [4], which is a widely used service orchestration platform. The Swarm Manager represents the cloud, while the Worker represents the edge. When the optimization module proposes a different service layout, Docker Swarm dynamically migrates the services based on the suggested layout.

We implemented our *resource manager module* using the Prometheus [30] monitoring system, which incorporates a time-series database to collect and store metrics from different services. Prometheus can collect metrics from multiple sources, such as those produced by cAdvisor (Container Advisor), which analyzes the resource usage and performance characteristics of running containers, and those generated by the services themselves using a Prometheus exporter. Prometheus can be asynchronously queried for metrics stored in the database.

4 Use-Case Driven Evaluation

We have evaluated the performance of `Shepard` by conducting two case studies for microservice deployment on the edge and have reported on two key performance metrics. Our results demonstrate that the dynamic placement of services,



(a) Battery percentage (b) Network outage

Fig. 4. `Shepard` notably enhances application availability: The graphs display node deaths and network connectivity for two distinct use-cases: energy and network-outage based deployments as seen in state-of-the-art methods [44]. Green regions mark times when the Heuristic-based node remains ACTIVE, contrasting with periods the static-based node is DEAD. Through dynamic service placement updates, `Shepard` effectively optimizes the energy budget.

facilitated by `Shepard`, allows for edge-assisted cloud infrastructure to achieve the optimal balance between cost, energy consumption, and latency while serving user requests from edge devices. Specifically, `Shepard` can achieve:

- up to 79% reduction in system downtime in a case study of solar-powered edge devices. This shows the effectiveness of `Shepard` in leveraging the underlying resources available. In this particular use-case, the energy forecast played a significant role in guiding the min-cut algorithm, among other parameters.
- up to 45% reduction in the total cost of deployment when compared to static deployment. For this use-case, the cost was an integral consideration for the min-cut algorithm, enabling cheaper computations at low latency.

4.1 Improving Service Availability

We conceive two case studies to show the improvement in micro-service based application's availability when equipped with `Shepard`.

Application. Similar to Farmbeats [44], we consider a scenario where microservices are deployed on an edge device for performing precision agriculture. This deployment comprises multiple services performing heterogeneous tasks such as collecting readings from sensors, creating summaries of sensor readings, and controlling a drone to get aerial imagery for creating precision maps from sparse

Table 1. Moveable/Non-moveable Micro-services

Services	Moveable
Orthomosaic creation	✓
Cow monitoring	✓
Storage monitoring	✓
Drone path planning	✓
Drone control	x
Aerial imagery compression	✓
Sensor reading	x
Web interface	✓
Crop suggestions	✓
Cross-farm analytics	x
Precision map creation	✓
Long-term storage	x
Weather forecast	✓

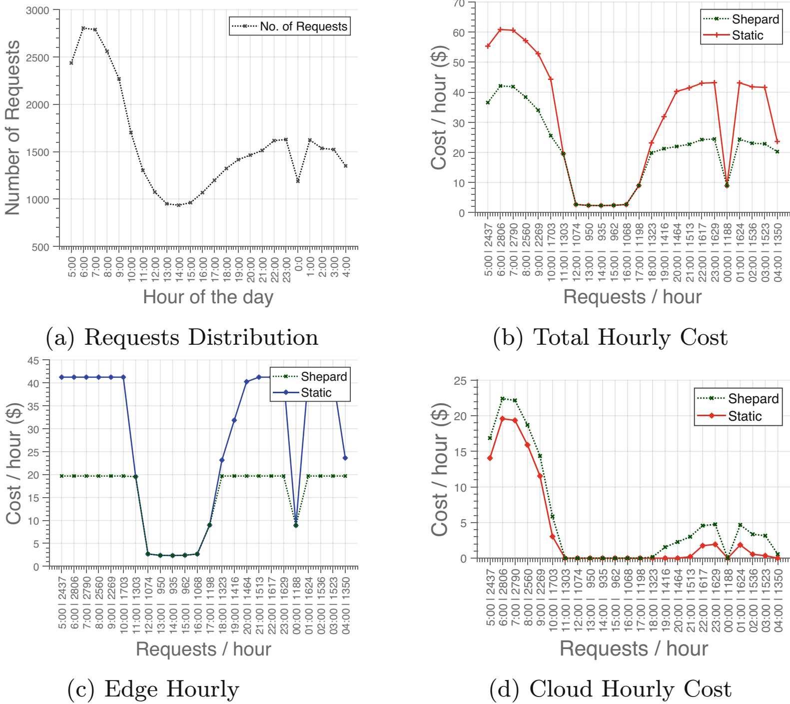
sensors deployment. Table 1 summarizes the services that can or cannot be moved between the edge device and the cloud.

Energy-Aware Dynamic Deployment. We first consider a scenario that takes into account the energy available to the device to perform its tasks.

Use-case. In this scenario, we assume that all services are hosted on a solar-powered edge gateway. Our system uses existing weather data to create a forecast model that predicts node failure based on the current deployment strategy. To do this, we count the consecutive number of hours where hourly energy consumption exceeds the energy gained using solar power. We use the same model for both deployment strategies, and Fig. 4a shows the comparison of system availability in all cases.

Results. As shown in Fig. 4a, a static deployment consumes energy at a very high rate compared to the dynamic deployment. This high energy consumption rate stems from the inability to move services dynamically from edge to cloud when the system can predict a power failure. In contrast, `Shepard` conserves energy by adapting its placement strategy and relocating services to the cloud, ensuring the node remains operational for a longer duration on the given battery charge.

Note that solar efficiency depends on the weather's cloudy nature and varies significantly throughout the day. This affects the strategy employed by the dynamic model, resulting in a different rate of energy consumption. Whenever the input energy is sufficient for the device to charge its battery, the number of services hosted on the edge device increases, increasing the device's energy consumption, as shown in Fig. 4a.



(a) Requests Distribution

(b) Total Hourly Cost

(c) Edge Hourly

(d) Cloud Hourly Cost

Fig. 5. Shepard reduces cost of deployment: By enabling energy- and resource-aware deployment, Shepard allows the requests to be serviced at the edge for as long as the cost of computing on the edge is less than that of the cloud. This protects edge devices from over utilization and reduces deployment cost.

Network Outage. We employ the same application [44] in a different use-case scenario to assess system downtime in the event of network failure caused by harsh weather conditions. It's essential to note that the weather conditions and parameters for this evaluation vary from the earlier energy-aware dynamic deployment assessment, offering insights into the system's adaptability under different challenges.

Use-Case. Harsh weather conditions can cause the network to become unavailable for extended periods [13, 17, 20]. This demands dynamic reassignment of services on the edge device to maintain network availability. Therefore, we develop a network outage model to predict network failure in harsh weather environments. This model forecasts future network availability, which helps Shepard determine when and which service to deploy on the edge. Figure 4b illustrates a comparison of system availability for a given network outage prediction.

Results. Our results demonstrate that whenever there is a network outage, Shepard can sustain it due to its dynamic nature. It reassigns all services to the edge devices if a predicted network failure occurs due to harsh weather

conditions. This dynamic reassignment enables all requests to be processed on the edge, thereby reducing application latency and increasing availability, as shown in Fig. 4b.

Conclusion. The dynamic reassignment of services in the precision agriculture scenario enables the device to operate during network outages and brings energy efficiency, which plays a critical role in maintaining the availability of an edge-assisted cloud infrastructure. Increased availability reduces application latency, a significant issue for cyber-physical systems with real-time constraints. Therefore, the findings presented emphasize the significance of our dynamic service deployment system in enhancing system efficiency and reliability.

4.2 Reducing Deployment Cost

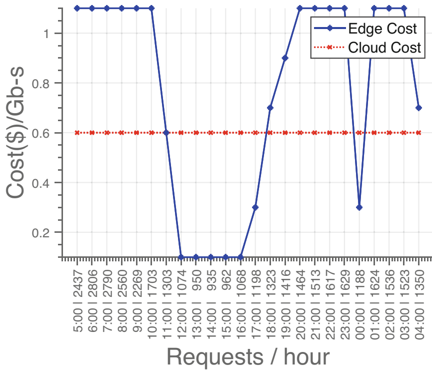
To analyze the impact of `Shepard` on the overall computation cost, we consider data from ride-hailing services such as Uber [41] and Lyft [22].

Application. When a user searches for a suitable ride nearby, multiple services are invoked before returning the final result. These tasks include finding the shortest path between the source and destination, calculating the fare, and more. As this is a distributed application, all these tasks must be performed in a central remote location. With a large user base, this application can put a significant load on the cloud for computation. Additionally, the distance between users and the cloud can introduce communication latency, which can negatively impact the user experience. Thus, an edge device near the user capable of performing the same computation as the cloud can help reduce the cloud's latency and load. However, the introduction of such a device increases the overall cost of the cloud.

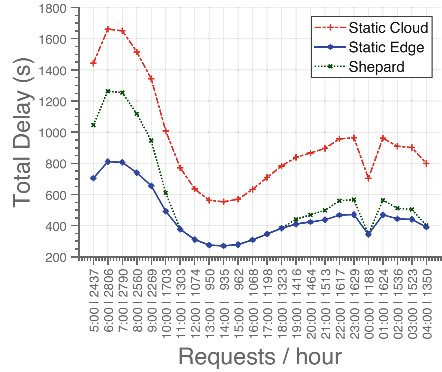
Setup. We use 24 h of Uber movement data [42] to determine the number of ride requests sent every hour. The extracted data is shown in Fig. 5a. We consider the computation cost of the cloud and edge as used in existing literature [46]. Moreover, we compare `Shepard`'s dynamic service placement model against a *static* baseline service placement approach where all requests are sent to the edge until it reaches 100% utilization, after which every request is directed towards the cloud.

Impact on Cost. To better understand the cost impact of `Shepard`, we compare the cost of computing on a system equipped with `Shepard` against a static approach where a request is redirected to the cloud only when the edge is being utilized to its full potential.

Edge vs. Cloud Cost. Each request serviced at the cloud costs the same, regardless of the rate at which requests are received. However, the cost of computation changes with the number of requests being serviced by the edge. The higher the utilization, the higher the cost of performing computations at the edge, as shown in Fig. 6a.



(a) Edge vs Cloud Cost Distribution



(b) Total Delay

Fig. 6. Deployment cost on edge vs cloud. Shepard can find the sweet spot: For higher number of requests, the cost of computing on the edge is greater than of the edge. Shepard reduces this cost by shifting the load to the cloud while minimal increase in latency.

Edge Cost per Hour. Edge devices are charged an extra amount if their utilization crosses a certain threshold. Static deployment techniques fail to cater to such requirements and utilize the edge to its full potential, moving service to the cloud only when the edge device's utilization reaches 100.

Figure 5c shows the cost of performing computations on the edge device for every hour of the day. When compared with Shepard's deployment, we can observe that the cost of computing on the edge is very high for static deployment whenever the number of requests is high. Unlike clouds, the cost of computing on an edge device depends on the current load; the higher the load, the higher the price. Static deployments utilize the edge device to its full potential, thereby performing computations on the edge during peak hours for a longer duration of time.

Cloud Cost per Hour. The cost of computation of a static deployment is meager at the cloud end of the edge-assisted cloud infrastructure. Since most requests are handled at the edge end, the number of requests serviced by the cloud is very low compared to Shepard. As a result, the cloud's computational cost is low compared to dynamic deployment, as shown in Fig. 5d.

A comparison of the total cost incurred in both scenarios is given in Fig. 5b. Although Shepard's dynamic placement requires more cost for computations, the overall cost of computations for edge-assisted cloud infrastructure is still 20% to 30% lower compared to static placement due to the savings in cost at the edge, as shown in Fig. 5c.

Impact on Latency. While Shepard's dynamic placement reduces cost, it is important to note that placing services on the cloud incurs an additional trans-

mission and propagation delay on every request, resulting in increased application latency. To measure the effect of `Shepard`'s placement strategy on latency, we used data for all trips lasting 30 min in the Uber movement dataset [42].

We calculated the total delay of all requests in every hour and performed these calculations for three different scenarios:

1. Static Cloud: when all requests are serviced from the cloud.
2. Static Edge: when all requests are serviced from the edge.
3. Dynamic Delay: when requests are serviced using `Shepard`'s service placement.

Figure 6b shows the total delay for all three approaches. We can see that `Shepard`'s delay lies between the two static methods, as the static approach cannot adapt to the changing request patterns. By adapting to user requests, `Shepard` always deploys some services on the cloud to find the right balance between cost and latency. Scenarios 1 and 3 are two extreme cases where computations are performed either on the edge or the cloud. While performing computations on edge devices decreases latency, it significantly increases cost, rendering edge-assisted cloud infrastructure infeasible. On the other hand, servicing all requests on the cloud significantly increases the latency at a minimal cost reduction. `Shepard` finds the sweet spot between these two extremes to combine the best of both worlds.

Conclusion. Unlike static approaches, `Shepard` dynamically places services on the edge and cloud according to the changing needs of the application. This allows for minimization of edge utilization during peak hours, resulting in cost reduction. At the same time, `Shepard` carefully chooses the set of services to be deployed on the edge so that application latency remains as low as possible.

5 Related Work

A large body of work investigates service placement in the cloud computing domain. However, to scale down our discussion on related works, we focus on dynamic service placement in edge-assisted cloud infrastructure. We broadly divide related literature into two categories: computational offloading in edges and service orchestration strategies.

5.1 Computational Offloading in Edges

Du et al. [6] have formulated the computational offloading and content caching problem as a joint optimization problem to enhance the operator's profit in queuing cellular network. Samanta et al. [34] have proposed an adaptive service offloading scheme for the mobile edge computing platform in the presence of multiple edge devices to maximize profit and reduce latency. Liu et al. [21] have proposed a game-based approach to model the interaction between the edge cloud and users. The edge cloud sets prices to maximize its revenue while

ensuring that its finite computation capacity is not exceeded. For the given prices, each user locally makes offloading decisions to minimize its own cost, which is defined as latency plus payment.

Zhang et al. [47] have used unmanned air vehicles (UAVs) to assist vehicle-carried terminals in performing complex computational tasks. These tasks can be offloaded onto these UAVs, which would otherwise have been offloaded to the cloud, thus causing long delays. Zhang et al. have formulated an energy-aware optimal resource allocation problem to maximize the utility under an edge-assisted cloud computing environment. Zhu et al. [48] have proposed a cooperative mobile edge computing model to reduce both energy consumption and task execution latency, and have formulated a convex optimization problem to utilize energy efficiently. Li et al. [19] have proposed a similar approach that reduces energy consumption and access delays while deciding the service placement scheme.

The primary focus of these works is on minimizing price and energy consumption. However, focusing only on these two factors is of little benefit if we cannot fit the selected service on a resource-constrained edge device. In contrast, `Shepard` not only minimizes price and energy but also considers the resource constraints of the embedded edge devices, such as memory and energy buffer. These constraints play an essential role in deciding the set of services to be hosted on the cloud and edge.

5.2 Service Orchestration Strategies

Hu et al. [12] propose a binary partitioning algorithm based on a randomized contraction algorithm to reduce inter-service traffic in cloud computing environments. Kiss et al. [15] propose a framework that automatically adjusts the supply of cloud services based on application demands. This framework enables developers to integrate cost and performance optimization mechanisms into their application code using APIs, allowing for dynamic orchestration of services in the cloud computing environment. Selimi et al. [37] utilize the state information of the underlying network to devise a service placement strategy for micro-cloud infrastructure that reduces bandwidth usage and latency for end-users.

Farhadi et al. [9] propose a two-time-scale solution for joint service placement and request scheduling in edge clouds under communication, computation, and storage constraints. Petri et al. [29] propose an orchestration engine that can be located at the network edge and supports the allocation of tasks to edge and/or cloud resources for coordinating complex industrial processes. Using the proposed orchestrator, they have achieved lower task completion times while reducing data transfer times. Sampaio et al. [35] propose a platform-independent runtime adaptation mechanism to reconfigure the placement of microservices based on their communication and resource usage.

While many of these works share similarities with ours in spirit, our approach is unique. We focus on edge-assisted cloud infrastructures, where edges are resource-constrained devices that represent a valuable computational resource. Our system provides an efficient way of finding the optimal service placement

strategy while considering the resource-constrained nature of edge devices. In contrast, these approaches focus on either the cloud computing paradigm or propose adaptations applicable to service orchestration in general.

6 Conclusion

In conclusion, `Shepard` offers a novel approach to microservice placement in edge-assisted cloud infrastructures, leveraging a labeled-graph representation to efficiently find the best graph-cut while considering network bandwidth and edge device constraints. By using heuristics to aggregate edges and vertices of the graph, `Shepard` can effectively scale down the problem of finding the cut, resulting in a significant impact on performance. As demonstrated, `Shepard` can increase service availability in solar-powered edge deployments by leveraging future weather prediction to relocate services, and also reduce the overall cost of service deployment in ride-hailing applications without affecting performance. This highlights the versatility and potential of `Shepard` as a solution for microservice placement in resource-constrained edge-assisted cloud infrastructures. Future work could focus on exploring the effectiveness of `Shepard` in other use cases and evaluating its performance in more extensive deployments.

References

1. Al Shayeji, M.H., Samrajesh, M.: An energy-aware virtual machine migration algorithm. In: 2012 International Conference on Advances in Computing and Communications, pp. 242–246. IEEE (2012)
2. Amazon: Microservices (2019). https://aws.amazon.com/microservices/
3. Beloglazov, A., Buyya, R.: Energy efficient allocation of virtual machines in cloud data centers. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 577–578. IEEE (2010)
4. Docker: Swarm mode overview (2019). https://docs.docker.com/engine/swarm/
5. Docker: What is a container? (2019). https://www.docker.com/resources/what-container
6. Du, J., Zhao, L., Feng, J., Chu, X., Yu, F.R.: Economical revenue maximization in cache enhanced mobile edge computing. In: 2018 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2018)
7. DZone: Enterprises are adopting microservices architectures (2019). https://dzone.com/articles/new-research-shows-63-percent-of-enterprises-are-a
8. DZone: Lessons from the birth of microservices at google (2019). https://dzone.com/articles/lessons-from-the-birth-of-microservices-at-google
9. Farhadi, V., et al.: Service placement and request scheduling for data-intensive applications in edge clouds. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pp. 1279–1287. IEEE (2019)
10. Foundation, O.: Node.js (2019). https://nodejs.org/en/
11. Ghorbani, S., Godfrey, B., Ganjali, Y., Firoozshahian, A.: Micro load balancing in data centers with drill. In: Proceedings of the 14th ACM Workshop on Hot Topics in Networks, p. 17. ACM (2015)

12. Hu, Y., de Laat, C., Zhao, Z.: Optimizing service placement for microservice architecture in clouds. Appl. Sci. **9**(21), 4663 (2019)
13. Ismail, B.I., et al.: Evaluation of docker as edge computing platform. In: 2015 IEEE Conference on Open Systems (ICOS), pp. 130–135. IEEE (2015)
14. Jaramillo, D., Nguyen, D.V., Smart, R.: Leveraging microservices architecture by using docker technology. In: SoutheastCon 2016, pp. 1–5. IEEE (2016)
15. Kiss, T., et al.: Micado-microservice-based cloud application-level dynamic orchestrator. Future Gener. Comput. Syst. **94**, 937–946 (2017)
16. Kord, N., Haghighi, H.: An energy-efficient approach for virtual machine placement in cloud based data centers. In: The 5th Conference on Information and Knowledge Technology, pp. 44–49. IEEE (2013)
17. LeClair, D.: The edge of computing: it's not all about the cloud. Inov. Insights (2014)
18. Lewis, G.A., Echeverría, S., Simanta, S., Bradshaw, B., Root, J.: Cloudlet-based cyber-foraging for mobile systems in resource-constrained edge environments. In: Companion Proceedings of the 36th International Conference on Software Engineering, pp. 412–415. ACM (2014)
19. Li, Y., Wang, S.: An energy-aware edge server placement algorithm in mobile edge computing. In: 2018 IEEE International Conference on Edge Computing (EDGE), pp. 66–73. IEEE (2018)
20. Liu, K., Manangi Ravindrarao, N., Gurudutt, A., Kamaal, T., Divakara, C., Prabhakaran, P.: Software-defined edge cloud framework for resilient multitenant applications. Wirel. Commun. Mob. Comput. **2019** (2019)
21. Liu, M., Liu, Y.: Price-based distributed offloading for mobile-edge computing with computation capacity constraints. IEEE Wirel. Commun. Lett. **7**(3), 420–423 (2017)
22. Lyft: Lyft (2019). https://www.lyft.com/
23. Mazrekaj, A., Shabani, I., Sejdiu, B.: Pricing schemes in cloud computing: an overview. Int. J. Adv. Comput. Sci. Appl. **7**(2), 80–86 (2016)
24. Meiklejohn, C., Miller, H., Lakhani, Z.: Towards a solution to the red wedding problem. In: {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 2018) (2018)
25. Nastic, S., Truong, H.L., Dustdar, S.: Data and control points: a programming model for resource-constrained IoT cloud edge devices. In: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 3535–3540. IEEE (2017)
26. Nginx: Adopting microservices at Netflix: Lessons for architectural design (2019). https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/
27. Nginx: The future of application development and delivery is now (2019). https://www.nginx.com/resources/library/app-dev-survey/
28. Oberheide, J., Veeraraghavan, K., Cooke, E., Flinn, J., Jahanian, F.: Virtualized in-cloud security services for mobile devices. In: Proceedings of the First Workshop on Virtualization in Mobile Computing, pp. 31–35. ACM (2008)
29. Petri, I., Rana, O., Zamani, A.R., Rezgui, Y.: Edge-cloud orchestration: strategies for service placement and enactment. In: 2019 IEEE International Conference on Cloud Engineering (IC2E), pp. 67–75. IEEE (2019)
30. Prometheus: Prometheus (2019). https://prometheus.io/
31. Qayyum, F., Naeem, M., Khwaja, A.S., Anpalagan, A., Guan, L., Venkatesh, B.: Appliance scheduling optimization in smart home networks. IEEE Access **3**, 2176–2190 (2015)

32. Ramachandran, G.S., Contreras, S.L., Krishnamachari, B., Kozat, U.C., Ye, Y.: Publish-pay-subscribe protocol for payment-driven edge computing. In: 2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 2019) (2019)
33. Ren, J., Yu, G., Cai, Y., He, Y.: Latency optimization for resource allocation in mobile-edge computation offloading. IEEE Trans. Wireless Commun. **17**(8), 5506–5519 (2018)
34. Samanta, A., Chang, Z.: Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint. IEEE Internet Things J. **6**(2), 3864–3872 (2019)
35. Sampaio, A.R., Rubin, J., Beschastnikh, I., Rosa, N.S.: Improving microservice-based applications with runtime placement adaptation. J. Internet Serv. Appl. **10**(1), 4 (2019)
36. Scott, J.A.: A Practical Guide to Microservices and Containers. Addison-Wesley, Reading (1972)
37. Selimi, M., Cerdà-Alabern, L., Sánchez-Artigas, M., Freitag, F., Veiga, L.: Practical service placement approach for microservices architecture. In: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 401–410. IEEE (2017)
38. Singh, A., Korupolu, M., Mohapatra, D.: Server-storage virtualization: integration and load balancing in data centers. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, p. 53. IEEE Press (2008)
39. Singh, V.K., Dutta, K.: Dynamic price prediction for amazon spot instances. In: 2015 48th Hawaii International Conference on System Sciences, pp. 1513–1520. IEEE (2015)
40. Sorkhoh, I., Ebrahimi, D., Atallah, R., Assi, C.: Workload scheduling in vehicular networks with edge cloud capabilities. IEEE Trans. Veh. Technol. **68**(9), 8472–8486 (2019)
41. Uber: Uber (2019). https://www.uber.com/
42. Uber: Ubermovement (2019). https://movement.uber.com/
43. Vaquero, L.M., Rodero-Merino, L., Buyya, R.: Dynamically scaling applications in the cloud. ACM SIGCOMM Comput. Commun. Rev. **41**(1), 45–52 (2011)
44. Vasisht, D., et al.: FarmBeats: an IoT platform for data-driven agriculture. In: 14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 2017), pp. 515–529 (2017)
45. Vögler, M., Schleicher, J., Inzinger, C., Nastic, S., Sehic, S., Dustdar, S.: LEONORE–large-scale provisioning of resource-constrained IoT deployments. In: 2015 IEEE Symposium on Service-Oriented System Engineering, pp. 78–87. IEEE (2015)
46. Xu, H., Li, B.: Dynamic cloud pricing for revenue maximization. IEEE Trans. Cloud Comput. **1**(2), 158–171 (2013)
47. Zhang, L., Zhao, Z., Wu, Q., Zhao, H., Xu, H., Wu, X.: Energy-aware dynamic resource allocation in UAV assisted mobile edge computing over social internet of vehicles. IEEE Access **6**, 56700–56715 (2018)
48. Zhu, S., Gui, L., Chen, J., Zhang, Q., Zhang, N.: Cooperative computation offloading for UAVs: a joint radio and computing resource allocation approach. In: 2018 IEEE International Conference on Edge Computing (EDGE), pp. 74–79. IEEE (2018)