Smart Edge Service Update Scheduler: An Industrial Use Case

Sergio Moreschini[1], Francesco Lomio[1], David Hästbacka[1], and Davide Taibi[1,2]

[1] Tampere University, Tampere, Finland {name.surname}@tuni.fi
[2] University of Oulu, Oulu, Finland {name.surname}@oulu.fi

Abstract. Software systems need to be maintained and frequently updated to provide the best possible service to the end-users. However, updates sometimes, cause the system or part of it to restart and disconnect, causing downtime and potentially reducing the quality of service.
In this work we studied and analyzed the case of a large Nordic company running a service-oriented system running on edge nodes, and providing services to 270K IoT devices. To update the system while minimizing downtime, we develop a smart edge service update scheduler for a service-oriented architecture, which suggests the best possible update schedule that minimizes the loss of connections for IoT devices.
Our approach was validated by applying the scheduling algorithm to the whole system counting 270k edge nodes distributed among 800 locations. By taking into account the topology of the software system and its real-time utilization, it is possible to optimize the updates in a way that substantially minimizes downtime.

Keywords: Edge computing · provisioning · update scheduling · service-oriented · IoT.

1 Introduction

Software systems constantly need to be updated. Modern agile methods allow the continuous development and deployment of changes. However, the deployment of updates can require the restart of the system or part of it.

When considering widely used software systems, and in particular critical systems, downtime is usually unacceptable, and different strategies should be considered to avoid or minimize downtime as much as possible.

In our case, a very large Nordic company[3] is running a service-based system. The system is running 24/7 and it is deployed on edge and cloud, in multiple countries. Among different countries, the system has more than 800 locations, with an average of 336 edge nodes for each location for a total of $\sim 270k$ edge nodes. Each edge node provides a service to 100-1000 users connected simultaneously totaling 270 million connected Internet of Things (IoT) devices.

[3] For reasons of NDA, we are not allowed to disclose the name of the company, nor the low-level details of the use case.

The company is continuously developing the system using an agile methodology and the continuous building of the system needs to deploy a new version of the code every day. However, the deployment of the new version requires the restart of each location, taking an average of 30 minutes, and impacting all the edge nodes and related services provided to the connected IoT. During this time, all the end users connected to the edge nodes in the location, need to be rerouted to another edge node in a different location, to minimize the number of dropped service calls. However, the IoT devices can access only adjacent locations, due to the wireless technology adopted, increasing the complexity of the updates.

Given the daily upgrade time frame and the number of nodes, it would not be feasible to have a sequential upgrade schedule, as it would require more than 405 hours ($\sim$ 16 days).

Therefore, we intervened to support the company in identifying a smart update algorithm to schedule the updates of each location while reducing the number of service call drops as much as possible, maximizing the quality of service.

For this purpose, we defined a smart scheduling algorithm, validated it, and finally deployed it in production. The goal of the scheduling systems is to provide the suggested timing at which each location should start the provisioning process.

As a result, the company is now able to continuously deploy new updates, dropping only once a day, for 30 minutes, 20% of the calls to the service APIs.

The result of this work can be useful to researchers to validate the scheduling algorithm and to further extend it. Moreover, companies can benefit from this work by applying and extending it in production. It is important to remember that this algorithm is currently deployed in production, on a very large-scale system.

The remainder of this paper is structured as follows. In the next section, we introduce the necessary background and related work. In section 3 we introduce the smart edge service update scheduler, explaining its characteristics and its rationale. In section 4 we describe how the performance of the scheduling algorithm is measured. Section 5 includes the validation of the algorithm and the smart edge provisioning scenario. Section 6 finally presents our conclusions and draws future works.

2 Background and Related Works

2.1 Provisioning

The term provisioning is usually referred to as the process of preparing and equipping a software system to provide the best possible services to its users. However, since a system needs to be updated constantly, a vital part of the provisioning process is related to the updates and eventual restart of the applications. In this work, we use the term provisioning exactly to describe the update process of edge nodes (with consequent rebooting) to provide the best QoS to the system's users.

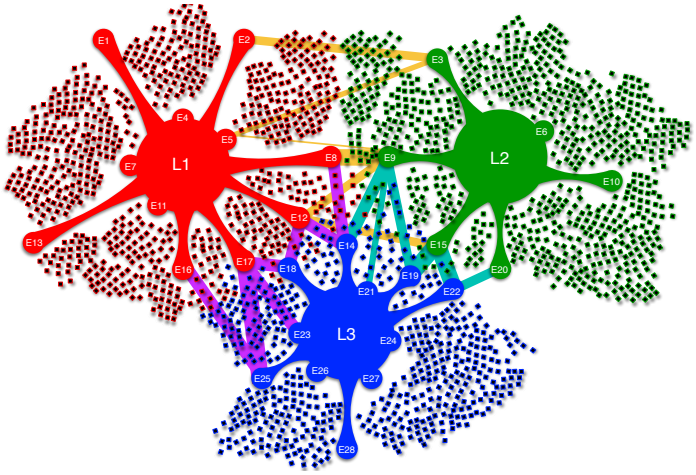

Fig. 1. Example of a system with multiple *locations* (L1, L2, and L3), each with multiple *edge nodes* (E1, E2, ..), and a variety of *IoT devices* connected to the the edge nodes of the closest location (squares with same colors as locations). Moreover, the lines connecting edge nodes of different locations indicate the possibility to handover the connections of the IoT devices.

2.2 Edge Computing

With the term Edge computing, we refer to a system where the computation is brought closer to the end user and the source of the data [1]. By keeping the majority of the data closer to the end users, there is a significant advantage in terms of lower latency and improved bandwidth compared to centralized systems. For this reason, whenever real-time processing is needed, edge computing allows bringing computation and data storage closer to the client.

2.3 Related Works

The increase in usage of edge technology and computing, including the proliferation of IoT devices, has increased the need for additional care needed to guarantee a sufficient quality of service. This system decentralizes the use of computational resources, bringing new issues in the management of the overall network.

Most of the research has therefore focused on how to optimize the provisioning of the resources for edge systems. Kherraf et al. [2], for example, proposed an approach that decomposes the resource provisioning and workload assignment into subtasks, allowing for higher performance trends in the overall system. Similarly, Cai et al [3] proposed a provisioning model called *edge federation*, which allows to schedule of resources among multiple edge infrastructure providers by characterizing the provisioning as a linear programming problem. Their method resulted in significantly reduced costs. Xu et al. [4] also tried to optimize the provisioning of resources in edge computing, by proposing a dynamic provisioning

method which, besides optimizing the resource scheduling, also tries to optimize energy consumption and the completion time. Another issue in resource provisioning is that sometimes it doesn't take into account edge-specific characteristics. Ogbuachi et al. [5] tackled this problem by integrating real-time information regarding physical, operational, and network parameters in the scheduling of 5G edge computing, showing that this approach improves the scheduling process compared to the default Kubernetes scheduler.

Among the works that tackled the resource provisioning process, some of them exploited machine learning models for optimizing it. Guo et al. [6], for instance, used a combination of Auto-Regressive Integrated Moving Average (ARIMA) and Back-Propagation Neural Network (BPNN), to predict the load and optimize the resource provisioning of an edge system. Similarly, Li et al. [7] used the same ARIMA and BPNN models to forecast the load and proposed a location for new requests to be filled, reducing the cost of provisioning.

3 The Scheduling Algorithm

The proposed smart scheduling algorithm is based on three different contribution factors, as described below.

- **Static Weight**: a factor relates to the information which is not going to change in the near future and, therefore, *static* in time. It is computed taking into account the topology of the system.
- **Dynamic Weight**: this factor, in contrast, includes all of the information which is not static in time and therefore related to throughput among different nodes. Specifically, in this model, the Dynamic weight is related to the number of active connections each location has at different time slots.
- **Cluster ID**: this factor assigns a value to each location showing which are similar and can be considered in the same cluster.

3.1 Static Weight

The topology of the network is presented as a structured file including the Edge Source and the Destination Edge for each different possible service. This means that between different couple of locations it is possible to have multiple edge-based connections. Moreover, we also know that different locations have a different number of edge nodes. As we believe that different information has different importance when impacting the topology of the network we computed the static weight s_w of each location as a weighted sum of this different information so that :

$$s_w = \alpha * W_E + \beta * W_L, \tag{1}$$

where α and β are two factors assigned to the different weights, W_E is the weight computed based on the number of edges within a single location and W_L is the weight computed based on how many connections there are between two different locations.

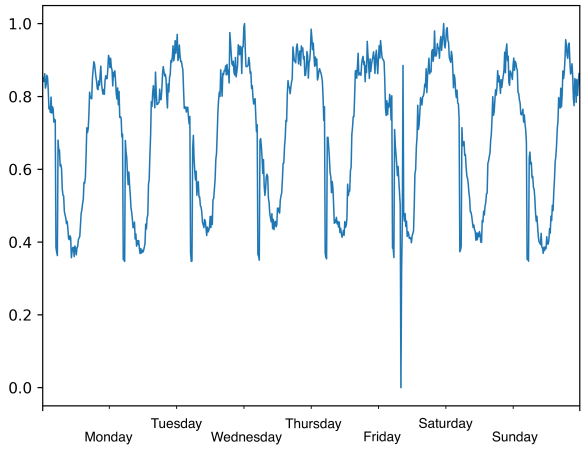

Fig. 2. Seasonality of the active traffic for a specific location.

More in particular we compute W_E as:

$$W_E(x_E) = \sum_{i=1}^{n} [l_i = x_E], \tag{2}$$

where l_i is the list of the unique edges, having the location as the prefix.

We compute W_L as:

$$W_L(x_E) = \sum_{i=1}^{n} [lS_i = x_E | lD_i = x_E], \tag{3}$$

where lS_i is the list of all source locations and lD_i is the list of all destination locations.

3.2 Dynamic Weight

The information related to the temporal evolution of the traffic for each location is especially useful in understanding which location to update (and therefore disconnect) first. The information is presented as a structured file that includes time series for each edge.

From an exploratory analysis of the aforementioned, the time series present an intra-day seasonality as well as a weekly seasonality (Figure 2). The objective of the algorithm used for this data is therefore to find the perfect time for each location that impacts the traffic the least.

In the specific, given a window of operation within the time series provided, for each location, we assign a weight based on how much provisioning would

affect the location. In our work, we divide the weights as optimal, suboptimal, acceptable, and irrelevant (i.e. 3, 2, 1, 0). This means that once we find the minima we assign to that specific time frame the value 3. Following, we compute the minima again, but this time within the time series having the previously time-frame dropped; we assign the value 2 to the newly found minima(s). We repeat the same procedure and we assign the value 1 to those that are found as 3rd minima. We assign 0 to the remaining time frames.

3.3 Cluster ID

A fundamental part of the algorithm is related to the process of clustering the different locations. To compute the clustering we rely on the python package NetworkX [8], used for the analysis of complex networks. NetworkX is mostly known in the literature for its ability to create a visual representation of a network, however, one of its less known but powerful strengths is the ability to compute cluster coefficients. In short, the cluster ID is the ID assigned to a location and used for grouping the locations sharing the highest number of edge connections between them.

The cluster coefficients have therefore been computed through the NetworkX Clustering function giving as an input W_L and the maximum number of allowable clusters. Once the coefficients are computed we created the clusters by computing evenly spaced areas and assigning each area based on the coefficients computed in the previous step.

3.4 Smart Edge Scheduling Algorithm

Given two files related to Topology (TN) and to the temporal evolution of the traffic (TS) we develop our algorithm as shown in Algorithm 1.

More in detail, for each of the possible i locations (l_i) in TN, we compute both the Edge-Based weight (W_E) and the location-based weight (W_L) as previously described in Equation 2 and 3 respectively. Once both of those are computed, we retrieve the static weight s_w for each possible location.

Then, by making use of W_L, we compute the cluster numbers using NetworkX.

Following, for each specific location, we assigned the dynamic weights DW by finding the minima (first, second and third) in TS. This means that the time frame with the minimum amount of data sent will have the highest dynamic weight assigned (3), the second minimum will have the second highest dynamic weight assigned (2), and so on until the weights are assigned.

Once the dynamic weights are assigned, we sort s_w from the lowest to the highest value. The reason behind this choice is that we want to prioritize locations that have the less amount of edges and connections as we will have fewer chances to redirect the connections to adjacent edges and therefore, impact more users.

Then for each location x_E in s_w we search for the maxima in DW, and most importantly, the time-frame ($TF(x_E)$) when the maxima in DW is found. Once

Algorithm 1: Smart Edge Provisioning Algorithm

```
for l_i in TN do
    W_E(x_E) = Σ_{i=1}^{n} [l_i = x_E];
    W_L(x_E) = Σ_{i=1}^{n} [lS_i = x_E | lD_i = x_E];
    s_w(x_E) = α + W_E(x_E) + β * W_L(x_E);
end
C = NetworkX(W_L)
DW = AssignDynamicWeights(TS)
SW = sort(sw)                          ▷ From lowest to highest value of x_E
for x_E in SW do
    TF(x_E) ← max(DW(x_E))
    if SC(C(TF)) is empty then
        SC(C(TF)) = TF(x_E)
    else
        TF2(x_E) ← max(DW(x_E), 2)
        if SC(C(TF2)) is empty then
            SC(C(TF2)) = TF2(x_E)
        else
            TF3(x_E) ← max(DW(x_E), 3)
            if SC(C(TF3)) is empty then
                SC(C(TF)) = TF(x_E)
            end
        end
    end
end
```

the $TF(x_E)$ has been detected we search if that specific TF has been assigned to any location within the same cluster C. If the TF for the specific cluster is vacant, then it is assigned to x_E, if not we repeat the same procedure for the second and third maxima. If all the possible detected TF have been already reserved, we move to the next location.

The reason for using TF is to maximize the degree of parallelism. We want to schedule inter-cluster parallel updates so that we have one update per node for each cluster, which means that the degree of parallelism depends the number of clusters created in the previous step.

Once all the locations have been served we have a clear schedule of which location should perform provisioning at each TF. On the other side, we will also have a list that reports which one is the correct TF to perform provisioning for each location. Inevitably, there are locations for which no suggested TF can be detected. This means that these locations (usually less than 5%), can be assigned to empty TF for their C without varying the impact.

4 Measuring the Scheduling Performance

To properly validate our algorithm it was fundamental for us to understand the performance of the model proposed. Moreover, it was important for us to

take into account some key factors such as the number of intra-edge connections broken while provisioning, and the amount of data lost in the same phase. For this reason, we created two metrics based on such factors: the intra-edge impact and the traffic impact.

4.1 Intra-edge impact

The first factor to take into account when measuring the performance of the network is the number of multiple connections between different locations. A fundamental part of the algorithm relies on the creation of clusters composed of locations that are strictly related to each other. The reason behind the choice is to reduce the number of parallel unavailable locations which share multiple connections. We know that different countries are composed of a different number of locations, therefore, countries with a higher necessity of connection are less demanding. For this reason, we need a factor that shows the ability of the proposed algorithm to keep dense active connections alive when the throughput is high and heavily penalize situations where suggested scheduling cannot be proposed.

The intra-edge impact takes as input the proposed scheduling and the topology of the network: the first provides information related to *when* a specific location is shut down, while the second about *which* connections are going to be impacted by the provisioning. To penalize a situation where scheduling was not possible, all the locations without a suggested schedule are grouped in the same time frame.

4.2 Traffic impact

The second factor to take into account when measuring the performance of the network is the amount of data lost during the provisioning. The goal of the algorithm is to minimize such an amount through optimal scheduling so that precise handovers can be performed and the chance of failure is reduced to the minimum.

In our environment, the information related to the traffic is provided in time frames of 15 minutes each. The provisioning time is set so that out of 30 minutes required, for the first 10 minutes (i.e. 1/3 of the time) the system runs at lower capability and tries to perform handovers, during the following 5 minutes the system is inaccessible, and for the last 15 minutes the location runs again at lower capability.

Knowing this, we try to schedule handovers during the whole provisioning time, however, we know from the literature [9] that usually 20% of handovers fail. When creating the traffic impact factor, we grouped all of this information and created a factor that takes as input the proposed scheduling and the temporal evolution of the traffic. This means that for each specifically scheduled provisioning in the first input we search a correspondence for the TF, once the traffic information is found we store it as $v1$ and the following TF as $v2$. Then,

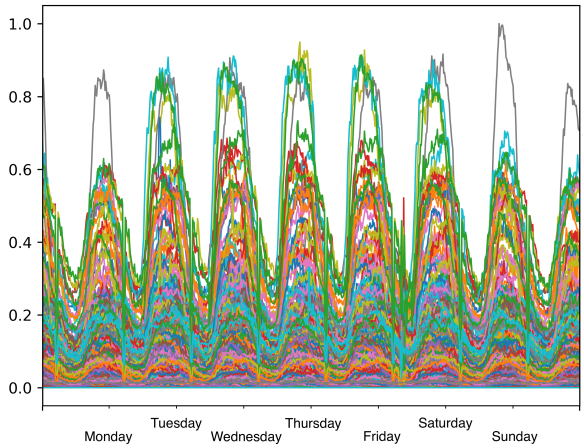

Fig. 3. Seasonality of the active traffic for all the locations considered.

we compute the traffic impact for the specific location as:

$$TI = \frac{\frac{1}{3} * v1 + (\frac{2}{3} * v1 + v2)/5}{5} \tag{4}$$

This means that we try to perform handovers all the time. However, statistically, 1 out of 5 times the handover fails and we need to compute the value we would lose in such an event. The traffic impact is composed of two parts, in the first we compute the complete outage which is taking one-third of the time of $v1$, in the second we compute the partial outage, which is taking two third of the time of $v1$, and the full $v2$. In our environment, during a partial outage the system can run at 80%, which means that during that period we lose 1/5 of the traffic.

5 Validation

To validate our scheduling algorithm, we used a system composed of 800 locations, with 270,000 edge nodes in total, averaging around 336 edge nodes for each location.

First of all, we calculated the static weights as described in 3.1. We obtained a value for each location; such value is only depending on the locations and the Edge devices, therefore, it is not changing, unless the architecture of the network itself would change.

Following, we calculated the dynamic weights described in 3.2. As it can be seen from Figure 3, there is a clear seasonality in the data, which allowed us and consider the temporal evolution of the system and therefore calculate the dynamic weights, for each of the locations.

Once the static and dynamic weights have been calculated, we calculated the cluster number using NetworkX as described in 3.3. This allowed us to have a modeled representation of the edge nodes and their location.

Our smart edge update scheduler produced therefore a proposed update scheduling. In Table 1 it is possible to see an example of the scheduling for some of the locations. As it can be seen, for each of the locations we have a time that represents the moment in which the update is scheduled.

Table 1. Update scheduling example.

Location	**Update Schedule**
235	2022-03-30 02:15:00 UTC
268	2022-03-30 02:30:00 UTC
224	2022-03-30 02:30:00 UTC
318	2022-03-30 02:45:00 UTC
362	2022-03-30 02:45:00 UTC
388	2022-03-30 02:45:00 UTC
402	2022-03-30 03:00:00 UTC
455	2022-03-30 03:00:00 UTC
469	2022-03-30 03:00:00 UTC

6 Conclusion

In this work, we provide a smart edge provisioning algorithm to minimize the number of dropped services and maximize the quality of service in a service-oriented architecture. We developed this algorithm to tweak at its best the environment provided resulting in a reduced amount of time necessary to perform a full upgrade of the elements composing the network and, therefore, not impacting availability of service in the hours with the highest demand. Such an environment is composed of a worldwide network divided into multiple locations, each one composed of multiple EDGE devices providing instruction to multiple IoT devices.

Ideally, the best possible outcome would be to update all the available locations without dropping any information provided to the IoT devices. This would mean that when performing the provisioning (update and restart), the amount of IoT devices connected to the location would be 0.

A possible way to achieve this condition would be by performing handovers to different locations. When performing handover we need to take into account two conditions:

– Network availability: when performing handover we need to make sure that the neighbor locations which are going to provide the service to the IoT devices will not be overpopulated by those. This would risk the malfunctioning of two locations.

– Failed handover: as described in Section 4.2, we know for sure that an average of 20% of handovers fail.

Therefore, it is very important to search for the time frame where each location has the minimum amount of throughput.

Given the latter as an input, we developed an algorithm tailored to this environment and impact values to validate it.

As the singular location-based provisioning would be impractical due to a large number of locations, our first goal has been to understand how to cluster different locations so that those could be updated in parallel. For this reason, we calculated weights based on the topology of the locations (and the edge nested in those) and the throughput of each location at each specific time frame.

By jointly analyzing the calculated weights we were able to understand which of these locations have more importance (some locations have fewer connections to other locations and therefore need to be carefully provisioned), and at what time there is less stream of information in the whole environment.

Our algorithm suggests schedules for most of the locations in the network. For some of them, it was not possible to provide an optimal schedule for two reasons:

– impact: the amount of information that they transmit is lower compared to other locations.
– overconnected: they have a high amount of connections to other locations and therefore the handover is easier

For these locations, the overall impact on the network between upgrading in different time frames can be neglected.

Acknowledgments

This work was partially supported by the Adoms grant from the Ulla Tuominen Foundation (Finland) and the MuFAno grant from the Academy of Finland (grant n. 349488).

References

1. M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
2. N. Kherraf, H. A. Alameddine, S. Sharafeddine, C. M. Assi, and A. Ghrayeb, "Optimized provisioning of edge computing resources with heterogeneous workload in iot networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 459–474, 2019.
3. X. Cao, G. Tang, D. Guo, Y. Li, and W. Zhang, "Edge federation: Towards an integrated service provisioning model," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1116–1129, 2020.

4. X. Xu, H. Cao, Q. Geng, X. Liu, F. Dai, and C. Wang, "Dynamic resource provisioning for workflow scheduling under uncertainty in edge computing environment," *Concurrency and Computation: Practice and Experience*, p. e5674, 2020.
5. M. C. Ogbuachi, C. Gore, A. Reale, P. Suskovics, and B. Kovács, "Context-aware k8s scheduler for real time distributed 5g edge computing applications," in *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, 2019, pp. 1–6.
6. J. Guo, C. Li, Y. Chen, and Y. Luo, "On-demand resource provision based on load estimation and service expenditure in edge cloud environment," *Journal of network and computer applications*, vol. 151, p. 102506, 2020.
7. C. Li, J. Bai, Y. Ge, and Y. Luo, "Heterogeneity-aware elastic provisioning in cloud-assisted edge computing systems," *Future Generation Computer Systems*, vol. 112, pp. 1106–1121, 2020.
8. D. A. Schult, "Exploring network structure, dynamics, and function using networkx," in *In Proceedings of the 7th Python in Science Conference (SciPy*. Citeseer, 2008.
9. P. Legg, G. Hui, and J. Johansson, "A simulation study of lte intra-frequency handover performance," in *2010 IEEE 72nd Vehicular Technology Conference - Fall*, 2010, pp. 1–5.