Dear IEEE Transactions on Services Computing – Editorial Board,

Thank you for your positive editorial view and the invitation to submit a revised version of the manuscript TSC-2023-10-0640, titled and authored:

**CROSS-EDGE ORCHESTRATION OF SERVERLESS FUNCTIONS WITH PROBABILISTIC CACHING**

Chen Chen, Manuel Herrera, Ge Zheng, Liqiao Xia, Zhengyang Ling, Jiangtao Wang

We are pleased to submit a revised version of our manuscript following careful incorporation of all the changes requested by the Reviewers. We also like to thank the Reviewers for their recommendations that have strengthened the manuscript overall. Please find below our point-by-point responses to Reviewers’ comments and a clear indication of the changes incorporated in the revised manuscript (also marked in yellow in the supplementary document). [...].

We are looking forward to your editorial decision.

Sincerely,

Dr Chen Chen (cc2181@cam.ac.uk)

On behalf of the rest of the authors

| **Reviewer Comment** | **Response** |
| --- | --- |
| **Reviewer #1** |  |
| 1. In the abstract: the phrase "on an as-used basis" is written in the past so it seems that the provision takes into account past needs. However, the correct idea is that resources are dynamically provisioned as needed. | We thank the reviewer for pointing this out. This is now amended as “dynamically provisions resources as needed” in the abstract. |
| 1. In the Introduction, page 1, second column, line 40. "responsible applications". Not sure if this is the intended word. I consider that by the use of such AI services your application does not become automatically responsible. | We agree with the reviewer and use the word “low-latency” instead. |
| 1. Page 3, first column, line 25: "Keep the above factors in mind, ..." -> Keeping the above factors in mind,...   Page 4, first column, line 37: "VNF". Explain acronyms the first time they appear (Virtual Network Function). | We agree with the reviewer, this is now amended. |
| 1. Page 4, second column, line 12: "In function caching, one function cannot serve multiple requests simultaneously while content caching can." This should be further explained. A function can server concurrently multiple requests or only one. For instance, when deploying a function in Knative the maximum number of concurrent invocations supported by the function can be specified in the yaml file. | Thank you. We agree that this sentence is not fully accurate. Our understanding is that the number of concurrent invocations is the number of requests routed to a container concurrently, the request may still be queued, waiting to be processed one by one. This depends on whether the container is multi-thread. For example, we can configure the concurrent invocations to 100, however, the container is single thread, so the requests are still processed one by one. Hence, we rephrase as “one function may not serve multiple requests simultaneously while content caching can”. |
| 1. Page 4, second column, line 51: "The decisions are then sent to the Kubernetes’ API server through HTTP requests." This is a bit confusing. You are deploying Knative but it seems that you are not using Knative at all. The developed scheduler communicates directly with the Kubernetes API thus you are not making use of Knative that has its own scheduler, its own autoscaler, etc. Therefore, why is the system deployed on top of Knative and not just on top of Kubernetes? Please clarify. | We developed a customised scheduler with our scheduling algorithm. Our scheduler only manages the four applications we proposed in table I. Our scheduler talks to the Knative resources through the Kubernetes API, in the same way Knative default scheduler orchestrates resources. The Knative default scheduler is still managing the rest of the system.  In this case, we could implement our algorithms without changing the existing knative system. We select this approach because we chat with the Knative community and they recommended this approach. To make this clearer to the readers, we further improve the explanation “The decisions are then sent to the endpoints of Knative resources through the Kubernetes' API server.” |
| 1. Page 4, second column, line 58: "a active" -> an active   Page 8, first column, line 35: "Size Size" duplicated word  Page 8, first column, line 39: "Frecuency Frecuency" duplicated word  Page 9: second column, line 34: "We implemented Knative over 11..." -> We deployed Knative over 11... | Thanks for pointing these out. These are amended. |
| 1. This is a suggestion for future work, maybe you can explore if you can really integrate your caching policy into Knative: https://pkg.go.dev/knative.dev/caching#section-readme | We are thankful to the reviewer’s suggestion. We are now looking at this document, it indeed looks highly relevant and promising. Hopefully we can take advantage of this functionality and even contribute to the project. |
|  |  |
| **Reviewer #2** |  |
| 1. Several recent studies have shown that different containers are likely to share layers, and thus hosting such containers on the same server could lead to reduced resource usage and improved performance. However, it seems that the proposed model does not consider these important factors. | We agree with the reviewer that layer sharing could reduce the startup time. However, layer sharing cannot resolve cold-start issues. Essentially, Layer sharing only partially reduces the startup time, images still need to be pulled from the registry and initialised, and hence the cold-start problem is not mitigated..  Nevertheless, in this paper, by using cached containers, we can avoid the startup time and mitigate the cold-start issue. Our understanding is that layer sharing is one approach to improve the latency, but it cannot avoid the cold-start problem, and thus it is not a fair comparison between function caching and layer sharing. |
| 1. The design insights behind the proposed algorithms are not explained very clearly, making the paper a bit hard to understand. | We reframe the algorithms with more details in section V.A and highlight it in the supplementary files. |
| 1. Although the ILP model is hard to solve when the problem scale is quite hard, how far would the performance of the proposed algorithm be from the optimal in small-scale tests? | We thank the reviewer for pointing this out. We prove the theoretical bound of performance in section V, A, page 7. By analysing the theoretical bound, we give a more comprehensive performance guarantee than comparing with the optimal in small-scale. Our understanding is that comparison to the optimal in small-scale tests usually appears in papers without analysis of theoretical bounds. |
|  |  |
| **Reviewer #3** |  |
| 1. The first issue is the use of the term / metric 'system cost.' The concept of system cost should be defined earlier in this paper (e.g., in the introduction)." | We agree and add now explanations in the abstract. That is, “the system cost (i.e., latency cost and container running cost)”. |
| 1. The system cost is essentially a linear sum of latency and container running cost. Can these two simply be summed? | Thanks for the great question. The system cost is a sum oflatency cost (not latency) and container running cost. Specifically, p\_n^v and d\_{v,v`} are the container switching cost and communication cost as shown in Table I, respectively. Those two costs are proportional to the corresponding latency. We use the system cost to feature tradeoffs between latency and container running. Latency can be converted to a penalty cost. We use the term system cost to indicate the performance of the system by jointly considering container running cost and the cost incurred by latency. Existing papers only consider latency or running cost which overlook the trade-off between them. For example, if the system aggressively caches containers without considering the running cost, the latency can be further reduced.  Vice versa, if the system only considers running cost, it could create containers only when necessary and ignore cold-start problems, however, the latency will go up remarkably. Actually, quite some paper uses the idea of latency cost:   * “Cost-Aware Big Data Processing Across Geo-Distributed Datacenters” - latency cost * “Traffic-aware and Energy-efficient vNF Placement for Service Chaining: Joint Sampling and Matching Approach” - network delay cost * “​​Orchestrating Virtualized Network Functions” - Penalty for SLO Violation |
| 1. Two baselines are LRU and FC. It would be beneficial to explain more about LRU and FC and how they are implemented. Specifically, LRU has many variations depending on the target systems, such as OS, distributed systems, and IoTs. The correctness of the proper LRU implementation is critical for justifying the evaluation results." | We thank the reviewer for pointing this out. We’ve adapted LRU and FC to our setup. For LRU, each container comes with a time tag that records the last time being used. Every time a container is used, the time tag is refreshed. Each edge node has a list of cached containers that are sorted by the time tag. Hence, the cache will evict a container which is least recently used when the space is insufficient. Similar to LRU, for FC each container comes with a tag that represents time to live. The cache will evict a container with a TTL equals 0 or the container with least TTL if the space is insufficient. We add some explanations in the manuscript, and the changes can be found in Section VI A, under performance benchmarks in page 8. |
| 1. Both the simulation and k-native implementation emphasized that the most important result was achieving low system costs. However, it is not easy to translate these numbers into real-world values. For instance, in Figs 4 and 7, the results are normalized, but the normalized target value remains unclear. | Low system costs indicate the cost incurred by latency and container running are low. In other words, the request is experiencing less latency and less container running cost to execute the service. This is common for example in paper “Online Orchestration of Cross-Edge Service Function Chaining for Cost-Efficient Edge Computing”, the system cost is normalised to indicate the overall performance of the system, justifying pCache mitigates the cold-start issue by reducing the system cost. By reducing the system cost, we observe that the cold-start frequency significantly decreases, implying that the end-to-end latency drops remarkably because the cold-start process could take several seconds which dominates the end-to-end latency.  The results are normalized against the max system cost achieved in the experiments, we agree with the reviewer and hence add explanation in the paper as “The system cost is normalized against the maximum system cost achieved in the experiments.” in page 9. |
| 1. A similar issue can be found in Fig. 5. For example, the system costs of LRU and FC are 59.7 and 125.6. Are these numbers actually significant enough to degrade QoS? | Thank you for raising the concern. The idea is to compare the overall system overheads by comparing the system cost. The numbers might not able to directly converted to QoS but we can observe that when system costs is high, the cold-start frequency is high and hence the Qos degrades because cold-starts could take several seconds. |
| 1. In Fig. 9, the cold-start latency of pCache is around 0.3. What does 0.3 represent? | The cold-start frequency represents the percentage of the requests that have a cold-start. Thus, 0.3 represents 30% of requests experience a cold-start when using pCache. To make this clear to readers, we add a sentence “The cold-start frequency is the percentage of requests that experience a cold-start in the experiment.” highlighted in page 8 first column. |
| 1. The correctness of implementing the simulator and simulation models is important. This is a single-blinded review, so it is ok to provide the source code (e.g., on Git) to allow reviewers to validate the correctness of the simulator implementation. | We agree with the reviewer, hence we provide the links here. We expect to document this in a better manner when we find time.  <https://github.com/LukasChenChen/serverless-cost-ns3.git>  Kantive implementation  <https://github.com/LukasChenChen/serverless-cost-k8s.git> |
| 1. The Azure dataset was used to generate serverless requests. However, it was collected in a data center / cloud-level FaaS setup. Why are cloud workloads considered sufficient to evaluate the performance in edge cases? It may be a good idea to use some IoT/CPS workloads to generate requests. | Yes, we agree with the reviewer. The reason is that we did not find any public serverless workloads for edge computing. Hence, we used Zipf distribution to generate the workloads between edge nodes which is widely used in a number of references such as “Service Placement with Provable Guarantees in Heterogeneous Edge Computing Systems”, “Latency-aware VNF chain deployment with efficient resource reuse at network edge”. |
| 1. The overhead of pCache is not reported. What is the time-complexity and space-complexity of pCache? Moreover, caching is essentially for better request scheduling. What happens if pCache employs a different scheduling algorithm like EDF with a deadline? | Thank you for pointing this out. We add the time-complexity in our paper in section V.C. Regarding the space complexity, the Knative master node usually comes up with sufficient memory. The applications are only run in worker nodes which means that the space complexity of the scheduler will not affect the container's performance. Hence, we hope the reviewer understands we tend to not analyse the space complexity.  Our understanding is that EDF would not suit this paper due to following reasons: (1) Our scheduling algorithm needs to consider the topology and connectivity between edge nodes because we wanted to reuse cached containers on the current edge node and the nearby edge node. However, EDF does not consider this. (2) Serverless requests usually do not come with a deadline. The requests are usually queued in a gateway and wait to be served one by one if containers are overloaded. |
| 1. Evaluation with k-native implementation may not be very practical. In edge computing, there are many interesting situations like communication failures (short-term/long-term), system/device-level failures, and resource-constrained issues like running out of memory. A more comprehensive evaluation would strengthen this work. | Yes, we agree with the reviewer that considering failures of nodes, communication and etc. would strengthen this work. Usually such papers will focus on the reliability or resilience of the system. We thank the reviewer and we would like to consider this in our future work.  We selected Knative because it is widely adopted by enterprises. Also, it enables the creation of a number of distributed nodes. We expect the reviewer will agree with us. Thank you. |
| 1. Lastly, as mentioned above, this work has some overlap with the authors' earlier work, 's-cache' [27] in EdgeSys '23. While s-cache is briefly mentioned in the related work, it would be beneficial to provide a more detailed explanation of the main differences and novelty of this work compared to s-cache. | We thank the Reviewer for this effort. It seems that both works have similarity because they all consider container placement in serverless edge computing. However, the difference is significant. This paper jointly considers latency and running cost while s-cache only considers latency. This paper uses a probabilistic caching method while s-cach’s approach is deterministic and based on priority. To summarise, the formulated problem and the proposed approach are different. Also, this paper includes simulation in a large-scale network while s-cache only conducted experiments in Knative. Summarising, we add explanations in the related work. “However, S-Cache does not consider the container running cost and aggressively caches containers with high priorities. “ |