Hindawi
Mobile Information Systems
Volume 2022, Article ID 4720114, 9 pages
https://doi.org/10.1155/2022/4720114

Research Article

A Node-Level Model for Service Grid

Yan Wang and Jifei Cai

College of Mechanical and Electrical Engineering, Beijing Institute of Graphic Communication, Beijing 102600, China

Correspondence should be addressed to Yan Wang; wangyanzi@bigc.edu.cn

Received 24 October 2021; Revised 20 November 2021; Accepted 27 November 2021; Published 6 May 2022

Academic Editor: Hye-jin Kim

This papers studies a high-performance node-level service grid model, which aims to solve the problem that the current pod-level service grid model affects the service operation and consumes many computing resources. The main method of the node-level service grid model is to improve pod-accompanied service grid sidecar with the node-accompanied service grid sidecar sharing of multiple pods, combined with the cut-through of user mode protocol stack and scaling of node-level service grid sidecar. By the performance comparison of pod-level service grid model and node-level service grid, we can conclude that node-level service grid model can isolate pod services without affecting service operation, significantly reduce memory consumption without multiplying with the number of pods, and largely reduce end-to-end network delay about 30% but the overall CPU consumption as the same as that of the pod service grid model. It indicates that the node service grid model can obtain better business benefits than the pod service grid model in container cloud, cloud service providers can provide grid services for more tenants with less memory resources and network latency, and adding grid services has no impact on the operation of user applications.

1. Introduction

Over the years, the concept of service-side architecture in the industry has been continuously evolving, from single modular architecture to SOA (service-oriented architecture), then to micro services, and finally to service grid [1]. The challenge of service grid is that the current grid model affects business operation and much computing resources with low performance [2–4].

1.1. Current Problems

(1) *Impact on Application Operation*. Sidecar of service grid will share pod computing resources with the application container. Because sidecar is not isolated and bound with the core, it will share the application CPU, resulting in 30%–50% performance degradation of the application.

(2) *Low Memory Resource Utilization*. Since sidecar of service grid is deployed in pod and the number of pod applications in the node is very large, there are hundreds and thousands of sidecars, which leads to the overall consumption of memory computing resources in the node. For example, since a sidecar takes 50 m memory, the sidecars of grid with 500 microservices will occupy total of 25 g memory.

1.2. Research Progress.

At present, some CPU and memory resource optimization work for sidecar of service grid has been carried out [5–9].

(1) *CPU Performance Optimization of Sidecar in Service Grid*. CPU performance optimization of sidecar in service grid mixer cache reduces the pressure of check on the server. Delta encoding and compress are optimized to reduce the network traffic generated by telemetry; web assembly bytecode format is used to realize the function of the original mixer in sidecar. These measures are helpful to the performance of the control plane, but they do not significantly improve the performance of the data plane. Sidecar inevitably encroaches on the CPU of the application container.

(2) *Memory Performance Optimization of Sidecar in Service Grid*. Isolating services by namespace and defining the communication relationship between services can reduce the number of listeners and clusters of sidecar so as to reduce the memory overhead of sidecar. Generally, the memory consumption can be reduced by 50%–70%, but for users, the service grid deployment is relatively complex. In the environment with strict resource requirements such as edge computing, the cumulative memory overhead of grid services is still too large.

1.3. Purpose of This Paper. To sum up, the performance optimization effect of the existing sidecar service relationship model based on POD deployment is limited. We hope to study a new sidecar relationship model between services to solve the impact of sidecar on application CPU operation and memory cumulative consumption and evaluate the effect of existing and new sidecar relationship models between services.

The rest of the paper is organized as follows.

The research model part introduces the current and new sidecar relationship models between services, the research method part introduces how the new model meets the elastic capacity expansion and interservice communication, the verification process part introduces how to build the verification environment, and the research results part gives the performance test results of CPU, memory, and delay under different relationship models, Finally, the conclusion gives the usability and benefit evaluation of the new model.

2. Research Models

2.1. Current Model. Service grid is an infrastructure layer used to handle interservice communication. Cloud native applications have complex service topologies, and the service grid guarantees that requests can travel reliably in these topologies. In practical applications, the service grid is usually composed of a series of lightweight network agents, which are deployed together with the application, but the application does not need to know their existence [10–18].

Currently, sidecar as a service grid agent is deployed in a pod and accompanied by the container (Figure 1). The CPU and memory performance problems of service grid hinder its large-scale applications.

The materials and methods section should contain sufficient detail so that all procedures can be repeated. It may be divided into headed subsections if several methods are described.

2.2. Our Model. To overcome the problems described in Section 1 of the current model, the main idea of this paper is to improve the service agent sidecar for the container in the pod to be shared by multiple containers in the node (Figure 2) [19–32].

(1) *Application Performance Improvement Ideas*. Because sidecar is independently deployed in a namespace in a node and can be isolated from the core, it will not affect the application CPU operation and occupy the application memory. Because the pod and sidecars are in different name spaces, we need to use cut-through technology to reduce communication delay. User mode protocol stack will get better communication performance than the kernel protocol stack.

(2) *The Idea of Improving the Utilization of System Resources*. Because sidecar is independently deployed in a namespace in node, the number of listeners and clusters in sidecar will be 1 to n (pod number), and the overall number of sidecar processes will be reduced and cores can be bound, so the memory consumption of sidecar will not increase with the increase in pod number, and the CPU efficiency will also be improved.

Compared with the pod-level sidecar model, the node-level sidecar model needs to solve two challenges caused by the change of sidecar deployment mode:

(1) When applying pod capacity expansion, how the shared deployed side car can synchronously expand and shrink the capacity as required by the application load is described in the elastic expansion section below.

(2) The sidecars applying pod and sharing deployment belong to different namespaces. How to efficiently communicate messages between sidecars and pod is described in the communication section below.

3. Methods of Our Research

3.1. Elastic Expansion. Sidecar initially produces a worker thread, which is a "nonblocking" event cycle, which monitors each listener, accepts new connections, instantiates the filter stack for each connection, and processes IO events in all connection life cycles.

All worker threads will listen to all listeners, and the protocol stack assigns the received socket to the worker thread. These threads also listen to the same socket and do not need to use spin locks for each connection. Once the worker accepts the connection, the connection will never leave that worker. All further processing is done within the worker thread.

Because the number of pod applications will change constantly, it needs the capacity of sidecar worker thread processing and pod number matching in the node. The method is that the sidecar worker thread supports elastic expansion as shown in Figure 3 [33].

(1) The elastic expansion process of sidecar is as follows:

    (1) With the increase in pod processing, the CPU load of worker threads has been created and gradually overloaded

    (2) After grid mng and ctl monitor that the sidecar worker exceeds the upper limit threshold, notify sidecar to expand its capacity

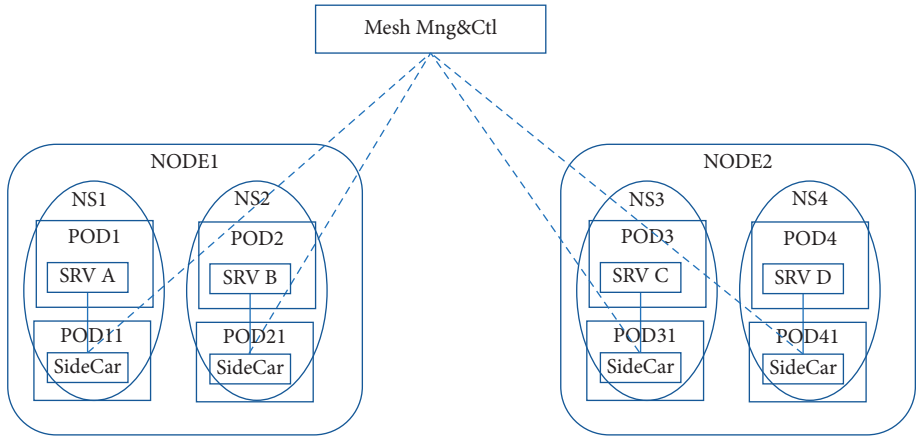

Figure 1: Pod-level one-to-one relationship model of sidecar.

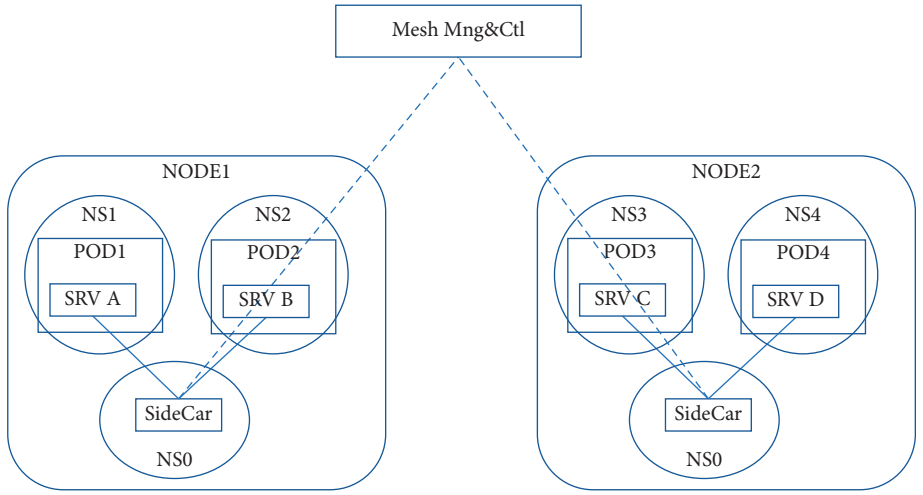

Figure 2: Node-level sharing relationship model of sidecar.

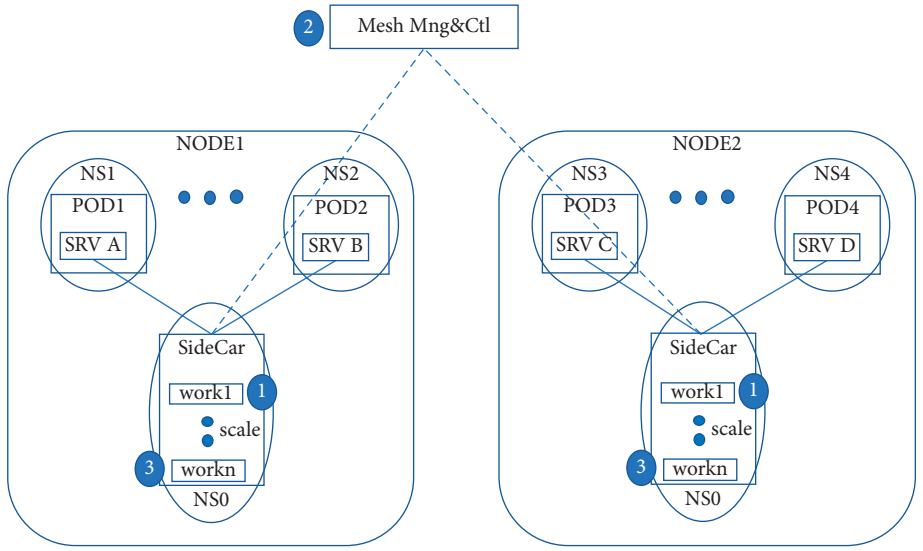

Figure 3: Elastic expansion of node sidecar.

(3) Sidecar adds new worker threads, and the worker thread with the lowest workload is preferred to accept connection processing

(2) On the contrary, the process of sidecar elastic shrinkage is as follows:

(1) With the decrease in pod load, the CPU load of the created worker thread is gradually lightened
(2) When grid mng and ctl monitor that the sidecar worker is less than the lower threshold, it notifies the sidecar to shrink
(3) Sidecar will not be scheduled to accept connection processing after processing the worker thread with the lowest load and then delete the shrink worker thread.

3.2. Communication. The communication scheme of sidecar intercepting the message of pod sender and receiver is shown in Figure 4.

To provide better performance when sidecar has a node-independent namespace, the cut-through mode of host stack in VPP user mode is adopted. This mode eliminates the copy of messages from user mode to kernel mode without going through the kernel stack and can also meet the zero copy of the shared memory of message queues in different namespace communication [34, 35].

3.2.1. Sender (Pod 1 Calls Pod 3). The message sent by pod 1 is aimed at pod 3 IP. The simple method is that node 1 uses the kernel IPTable NAT to intercept the message by redirection. However, since the message is to be transmitted from user state to kernel state to user state, there is low efficiency of twice message replication. Node-level sidecar adopts the efficient user state protocol stack cut-through mode, and the short connection between name spaces is completed in the protocol stack. Therefore, the redirection of the message sent out by the pod cannot be completed at L3 NAT level. The message interception needs to be completed at the protocol stack level. The method adopted is to hold the VCL connect initiated by the business pod of NS1 and redirect the destination address to the sidecar of this node IP, while VCL connect of NS0 sidecar does not intercept the hook, which is applicable to both TCP connections and UDP connections.

3.2.2. Receiver (Pod 1 Calls Pod 3). L3 forwarding of node 2 receives the message from node 1, sidecar 1 to pod 320.1.1.3 : 2 and then intercepts the message to 20.1.1.5 : 15001 of sidecar 2 of node 2 through L3 DNAT. For messages from sidecar 2 to pod 3, because the host protocol stack of NS1 and the host protocol stack of NS0 share the routing table, cut-through communication in different namespaces can be completed directly without any need for packet interception and redirection.

3.3. Verification Process. In this paper, we use a 2-node server to build the service grid cluster and build pod-level grid service software test environment and node grid service access test environment [36]. The software to be installed is as follows:

(1) Operating system Ubuntu LTS
(2) Container environment Docker CE
(3) Container management mik8s
(4) The energy of kernel protocol stack
(5) Enable VPP of user state protocol stack
(6) Fortio echo is applied to the client
(7) HTTP echo is applied to the server

The pod-level sidecar uses the Istio open-source project, the protocol stack is OS stack, and the host transceiver adopts the kernel IPTable. Node-level sidecar uses the sidecar VPP opensource project, the protocol stack is VPP host stack, and the host receiving adopts the user VPP.

Using fortio, a testing tool deployed in the client pod, it sends traffic to HTTP echo deployed in the service-side pod for a simple response. When pressure is applied to the application and service grid, the traffic running on it is within a controllable range, and a constant demand rate (RPS) is used to send the http request, which further increases HTTP connection to measure the response delay. By recording and comparing the throughput and delay of pod-level sidecar and node-level sidecar service grid cluster, the service impact and delay impact of different service grid relationship models are evaluated.

For resource consumption, the test tool is Istio-bench, which is deployed in the control node to measure CPU and memory consumption in sidecar of a service grid. Meanwhile, the CPU and memory consumption of the service grid of pod-level sidecar and node-level sidecar service grid are tested, respectively. The calculation resource consumption of different service grid relationship models is evaluated by this result.

Our test goal is to understand a node based on the service grid cluster environment. The performance of the service grid of sidecar requires that when the client application generates load pressure, the server application can meet the response within a fixed time range. We apply enough computing resources to the client and the server to avoid becoming a performance bottleneck. Gradually the pressure on the system is increased. The client accesses the pages served by the cluster. When the delay increases to a certain extent, the client will visit the pages served by the cluster and it is necessary to expand the node sidecar and reset the configuration parameters to complete the expansion in the test.

The pod-level service grid topology and node-level service grid topology are shown in Figures 5 and 6, respectively, which are used to compare the service grid impact on business and network delay and the CPU consumption and memory consumption used for comparing and evaluating the service grid.

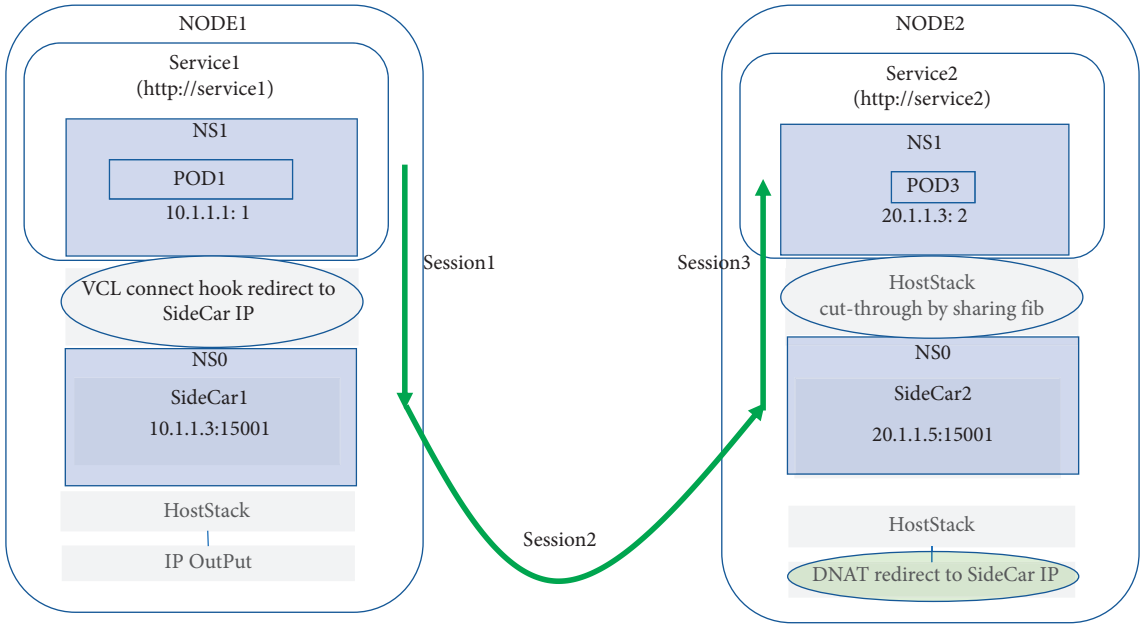

Figure 4: Sidecar intercepts the pod sender and receiver messages.

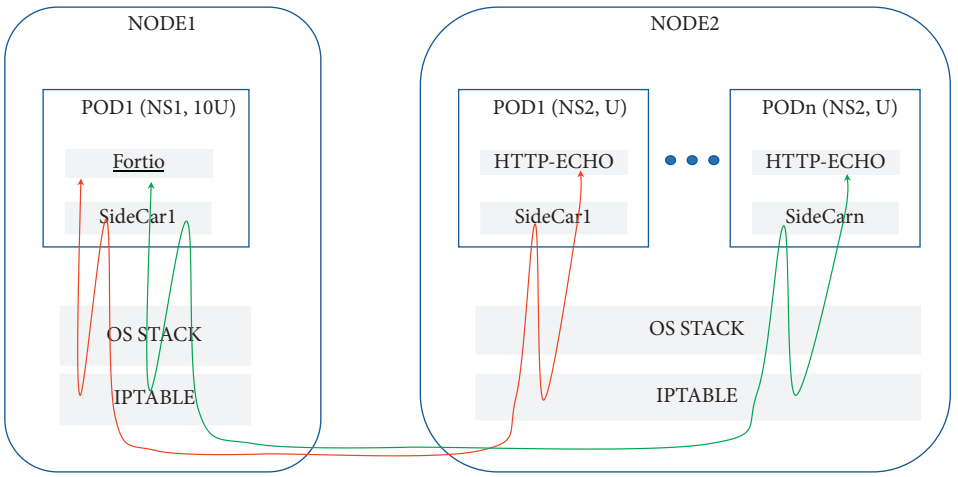

Figure 5: Pod-level service grid topology.

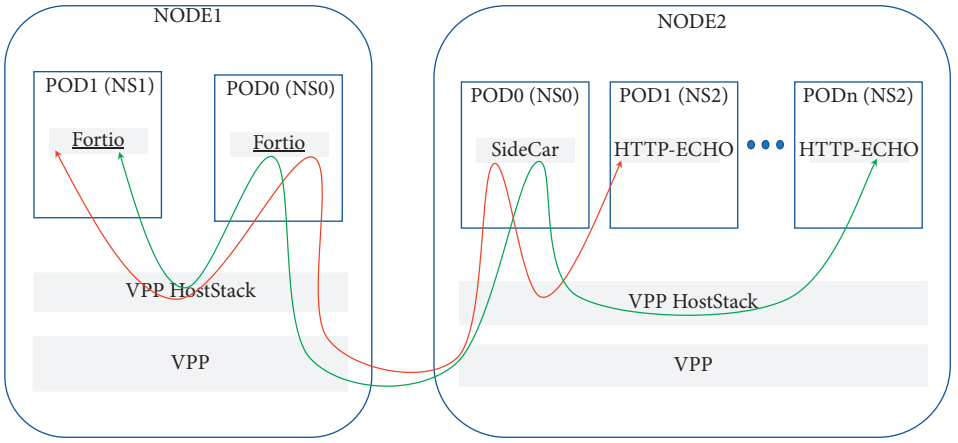

Figure 6: Node-level service grid topology.

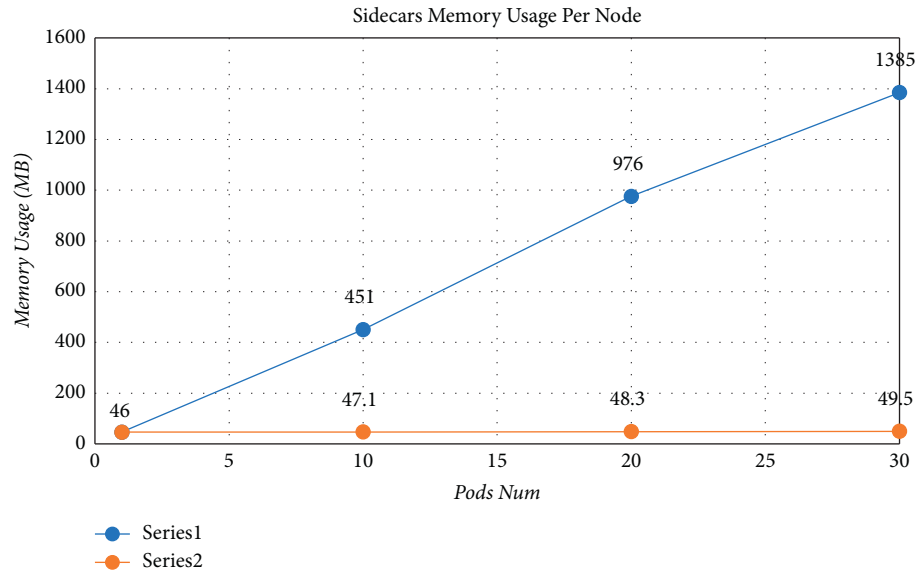

Figure 7: Memory consumption of different grid test topologies.

(1) *Comparative Evaluation Method of Memory Consumption in Service Grid*. First, the number of server pods is increased gradually, and the total memory consumption of sidecar in the server node of two topologies is observed and collected.

(2) *Service Grid CPU Consumption Comparative Evaluation Method*. Further, the fortio RPS of the client is increased gradually, and the total CPU utilization of sidecar in nodes of the server of the two topologies is observed and collected.

(3) *Service Grid Business Impact Comparative Evaluation Method*. Adding fortio RPS is continuted, and the maximum QPS throughput of HTTP echo in the node of the server end collecting the two topologies is observed.

(4) *Service Grid Network Delay Comparative Evaluation Method*. Then, fortio connections are added gradually under a certain RPS to observe and collect the network delay of the two topologies.

4. Research Results

4.1. Memory Consumption Comparison. The comparison test results of different grid topology memory consumption accessed between multiple pods are shown in Figure 7. Node grid service accesses more than pod grid service, and the service grid memory consumption is significantly reduced, and it no longer increases with the number of pods. Because there is only one client service and one server service, the node-level service grid only needs to consume listeners and cluster for these two services, and with the increase in the number of pods services, the cluster memory will increase, so the node-level service grid memory will increase a little, and no need to consume listeners and cluster memory for these two services as per pod-level service grid.

4.2. CPU Consumption Comparison. The comparison test results of CPU consumption of different grid topologies are shown in Figure 8. Node-level grid service is more accessible than pod-level grid service, and the total CPU consumption decreases slightly with RPS. This is mainly due to the independent deployment of node level, grid service, core isolation and reduction of grid service process number, and improvement of CPU scheduling performance. However, in general, the business process of grid service has not changed; therefore, the CPU performance improvement is limited, which is equivalent to the pod-level service grid model.

4.3. Service Impact Comparison. The test results of business impact comparison of different grid topologies are shown in Figure 9. Compared with pod-level grid service, node-level grid service does not occupy the CPU of the pod where the business is located and has no impact on the business operation. Therefore, the business can handle higher QPS throughput. This is mainly due to the fact that the node-level sidecar is independently deployed in a namespace and can be isolated from the core, so it will not affect the operation of the application CPU and occupy the application memory.

4.4. Network Delay Comparison. The comparison test results of different grid topology delays between two pods are shown in Figure 10. Compared with pod-level grid service, node-level grid service reduces the network delay between pods by about 30%. This is mainly due to the cut-through of user mode protocol stack and the receiving and sending of user mode host, which avoids the message replication from user mode to kernel mode and low-performance receiving and sending of kernel host of pod-level grid service.

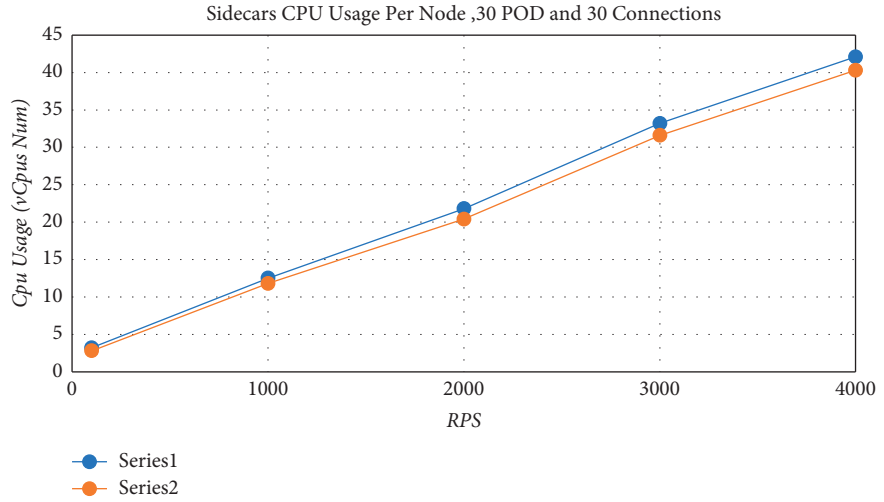

Figure 8: CPU consumption of different grid test topologies.

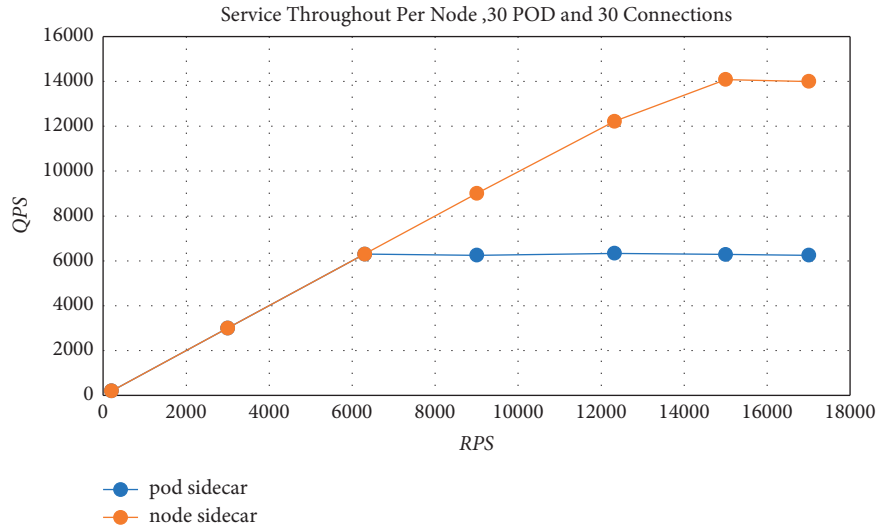

Figure 9: Service impact of different grid test topologies.

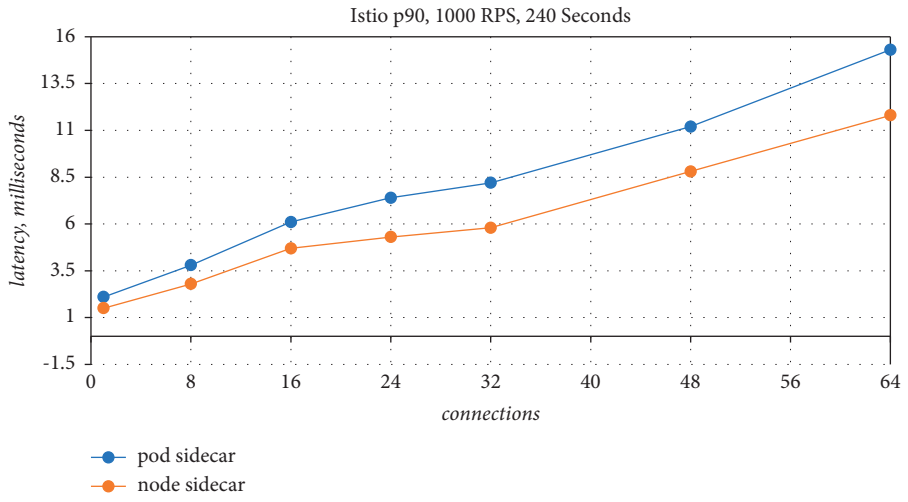

Figure 10: Network delay of different grid test topologies.

5. Conclusions

Service grid commercial scale needs to meet the requirements that the sidecar does not affect business operation and has lower consumption of computing resources to reduce user cost. Especially in the scenario of mixed cloud edge resource constraints, it is more necessary to ensure the lightweight of sidecar deployment. On the other hand, the end-to-end delay of the service grid is closely related to the performance of the underlying network. Compared with the kernel mode host, the user mode host has more advantages in the delay performance of message sending and receiving because it avoids the copy of messages between the user mode and the kernel mode. Service grid adopts a node-level sidecar model. Compared with the pod-level sidecar model because each node has only one sidecar, the number of listeners and clusters of the sidecar is reduced from n to 1, so the accumulated memory of the sidecar is significantly reduced, and the centralized processing of the sidecar also improves the CPU performance. On the other hand, compared with pod-level sidecar model, the node-level sidecar model also introduces some complexity, including that the node-level sidecar needs to be expanded with the load, and it is in a different name space from the service pod, so it needs to meet the lower delay of message sending and receiving through the interception and short circuit technology of user mode protocol stack.

From the previous research, it can be concluded that the node grid agent can isolate pod services without affecting business operation, significantly reduce the memory consumption to one, no longer multiply with the number of pods, and reduce end-to-end network delay by about 30%, and the overall CPU consumption is the same as the pod service grid, with a slight decrease of about 3%. It shows that the node-level sidecar model can get better application benefits than the pod-level sidecar model in container cloud service grid.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the Key Basic Research Project of Beijing Institute of Graphic Communication (EA202003) and General Project of Beijing Municipal Education Commission (KM201910015005).

References

[1] See https://www.nextplatform.com/2018/08/15/istio-aims-to-be-the-mesh-plumbing-for-containerized-microservices.

[2] See https://preliminary.istio.io/latest/docs/ops/deployment/performance-and-scalability/.

[3] R. Hentschel, C. Leyh, and A. Petznick, "Current cloud challenges in Germany: the perspective of cloud service providers," *Journal of Cloud Computing*, vol. 7, no. 1, pp. 1–12, 2018.

[4] M. B. Qureshi, M. M. Dehnavi, N. Min-Allah, M. Shuaib Qureshi, H. Hussain, and I. Rentifis, "Survey on grid resource allocation mechanisms," *Grid Computing*, vol. 12, no. 1, pp. 399–441, 2014.

[5] Y. B. Yang, "Research on Key Technologies of service grid performance optimization," *Computer applications and software*, vol. 38, no. 11, pp. 24–30, 2021.

[6] I. Kumara, J. Han, A. Colman, and M. Kapuruge, "Software-defined service networking: performance differentiation in shared multi-tenant cloud applications," *IEEE Transactions on Services Computing*, vol. 10, no. 1, pp. 9–22, 2017.

[7] B. Krašovec and A. Filipčič, "Enhancing the grid with cloud computing," *Grid Comput*, vol. 17, no. 1, pp. 119–135, 2019.

[8] Q. Ma, J. Y. Sun, and H. F. Li, "Research on data service optimization in high performance grid workflow," *Journal of Huazhong University of Science and Technology (Nature Science Edition)*, vol. 39, no. 1, pp. 15–18, 2011.

[9] Z.-G. Chen and B. Yang, "Task scheduling based on multi-dimensional performance clustering of grid service resources," *Journal of Software*, vol. 20, no. 10, pp. 2766–2775, 2009.

[10] See https://preliminary.istio.io/latest/about/service-mesh.

[11] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: migration to a cloud-native architecture," *IEEE Software*, vol. 33, no. 3, pp. 42–52, 2016.

[12] J. A. Valdivia, A. Lora-González, X. Limón, K. Cortes-Verdin, and J. O. Ocharán-Hernández, "Patterns related to microservice architecture: a multivocal literature review," *Programming and Computer Software*, vol. 46, no. 8, pp. 594–608, 2020.

[13] D. Salomoni, I. Campos, L. Gaido et al., "INDIGO-DataCloud: a platform to facilitate seamless access to E-infrastructures," *Journal of Grid Computing*, vol. 16, no. 3, pp. 381–408, 2018.

[14] A. M. Abdullah, S. M. Shamsuddin, F. E. Eassa, F. Saeed, and M. O. Alassafi, "Towards an intelligent framework for cloud service discovery," *International Journal of Cloud Applications and Computing*, vol. 11, no. 3, pp. 33–57, 2021.

[15] F. L. Zhang, "An intelligent contract micro service framework," *Journal of Software*, vol. 32, no. 11, pp. 3423–3439, 2021.

[16] J. T. Zhou and M. Wang, "Enterprise information integration framework for peer-to-peer semantic grid services," *Computer integrated manufacturing system*, vol. 16, no. 12, pp. 2697–2707, 2010.

[17] Z. Tao and Z. Q. Xiang, "Design and application of micro service architecture service mesh," *Automation technology and application*, vol. 39, no. 1, pp. 49–53, 2020.

[18] B. Wood, B. Watling, Z. Winn, D. Messiha, and Q. H. Mahmoud, "Remote method delegation: a platform for grid computing," *Grid Computing*, vol. 18, no. 1, pp. 711–725, 2020.

[19] J. S. Shi, J. B. Yuan, and F. S. Yuan, "Hierarchical grid trust model based on nearest service," *Journal of Nanjing University of Aeronautics & Astronautics*, vol. 43, no. 2, pp. 273–278, 2011.

[20] A. Haque, S. M. Alhashmi, and R. Parthiban, "Identifying and modeling the strengths and weaknesses of major economic models in grid resource management," *Grid Computing*, vol. 12, no. 1, pp. 285–302, 2014.

[21] M. Qian, Z. Liu, J. Wang, L. Yao, and W. Zhang, "Coordination-theoretic approach to modelling grid service composition process," *Journal of Systems Engineering and Electronics*, vol. 21, no. 4, pp. 713–720, 2010.

[22] G. P. Dai, "Cooperative game model for grid service composition task scheduling," *Journal of Beijing University of Technology*, vol. 38, no. 3, pp. 380–384, 2012.

[23] M. Selimi, L. Cerdà-Alabern, M. Sánchez-Artigas, F. Freitag, and L. Veiga, "Practical service placement approach for microservices architecture," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid '17*, pp. 401–410, Madrid, Spain, May 2017.

[24] K.-C. Wu, W.-Y. Liu, and S.-Y. Wu, "Dynamic deployment and cost-sensitive provisioning for elastic mobile cloud services," *IEEE Transactions on Mobile Computing*, vol. 17, no. 6, pp. 1326–1338, 2018.

[25] Y. R. Cui, M. C. Li, and J. He, "A grid node reputation evaluation algorithm and its application in the construction of service grid virtual organization," *Acta Electronica Sinica*, vol. 38, no. 7, pp. 1557–1562, 2010.

[26] Z. H. Wu and H. J. Chen, "From semantic grid to knowledge service cloud," *Journal of Zhejiang University - Science*, vol. 13, no. 44, pp. 253–256, 2012.

[27] C. Serven, J. Ejarque, D. Lezzi, and R. M. Badia, "Transparent orchestration of task-based parallel applications in containers platforms," *Journal of Grid Computing*, vol. 16, no. 1, pp. 137–160, 2018.

[28] Y. C. Yuan, X. P. Li, Q. Wang, and X. D. Zhang, "Grid workflow scheduling based on priority rules," *Acta Electronica Sinica*, vol. 37, no. 7, pp. 1457–1464, 2009.

[29] A. F. Qiu, "Design and implementation of disaster reduction service system combining micro service and middle platform concept," *Journal of Wuhan University (Natural Science Edition)*, vol. 45, no. 8, pp. 1288–1295, 2020.

[30] H. F. Li and Z. J. Xu, "Cascading fault prediction method in service grid," *Computer application and software*, vol. 38, no. 11, pp. 121–130, 2021.

[31] Y. Tian, H. Haitian, X. Xiaojian, and L. Bing, "Research on enterprise service governance based on service mesh," *Journal of Physics: Conference Series*, vol. 1673, no. 1, Article ID 012003, 2020.

[32] U. Paščinski, "QoS-aware orchestration of network intensive software utilities within software defined data centers," *Journal of Grid Computing*, vol. 16, no. 1, pp. 85–112, 2018.

[33] S. Draxler, H. Karl, and Z. A. Mann, "Joint optimization of scaling and placement of virtual network services," in *Proceedings of the 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 365–370, Madrid, Spain, May 2017.

[34] J. Y. Kun Meng, J. Cao, Z. Chen, L. Gao, and C. Lin, "Electricity services based dependability model of power grid communication networking," *Tsinghua Science and Technology*, vol. 19, no. 2, pp. 121–132, 2014.

[35] M. Selimi, L. Cerdà-Alabern, F. Freitag, L. Veiga, A. Sathiaseelan, and J. Crowcroft, "A lightweight service placement approach for community network micro-clouds," *Journal of Grid Computing*, vol. 17, no. 1, pp. 169–189, 2019.

[36] M. O. Keefe, J. Howard, and M. Jog, "Best Practices: Benchmarking Service Mesh Performance," *Istio form*, pp. 1–3, 2019, https://preliminary.istio.io/latest/blog/2019/performance-best-practices/.