Optimal Indexing: Supplementary Materials

Yuchen Sun, Lailong luo*, Deke Guo*, Li Liu, Junjie Xie

1 Complexity Analysis

Here, we provide more complexity analysis and summarize the results in Table 1. The results mainly contain two aspects: i) Computational Complexity including the time complexity and the space complexity, and ii) Network Costs including network latency, overlay hops, accessed servers and the packet size sent to the network.

A. Complexity Analysis for the Offline Stage. The offline stage mainly has three steps: (a) Edge Projection - project the edge servers into the virtual coordinates in the virtual space keeping the latency matrix, and then construct the forwarding graph; (b) Data Projection - project the data items into the virtual coordinates keeping the feature similarity; and (c) Data-Edge Association - determine each server's index range in the virtual space. Next, we analyze the complexity of each module.

a1) Complexity of LandmarkVC (Section 5.1.1): LandmarkVC generates the virtual coordinate for each landmark edge server. First, it needs to get the pairwise latency of all the landmark servers (*i.e.*, the latency matrix), and the latency matrix is sent it to a single server. Then, the server computes the virtual coordinates and return the results to each landmark server ($\mathcal{O}\left(|\widetilde{\Phi}|^2\right)$ packet sizes and $\mathcal{O}\left(|\widetilde{\Phi}|\right)$ accessed servers). Here we assume that any landmark server knows the network address of all the other landmark servers, so they can communicate directly through the network-layer routing protocol without multi-hop forwarding in the overlay network. Thus, the network latency is $\mathcal{O}(\tau)$ and the overlay hop is $\mathcal{O}(1)$. In Section 5.1.1, we provide two methods: Eigenvalue Decomposition relies on the eigenvalue decomposition of matrices, having $\mathcal{O}\left(|\widetilde{\Phi}|^3\right)$ time complexity, while the time complexity of Gradient Descent relies on the iteration number ξ is $\mathcal{O}\left(\xi d|\Phi|^2\right)$.

a2) Complexity of NormalVC (Section 5.1.1): NormalVC generates the virtual coordinate for each normal edge server. First, it needs to get the latency from the normal edge server to all the landmark servers (*i.e.*, the latency vector), and the latency vector is sent it to a single server. Then, the server computes the virtual coordinates and return the result to the normal server ($\mathcal{O}\left(|\widetilde{\Phi}|\right)$ packet sizes and $\mathcal{O}\left(|\widetilde{\Phi}|\right)$ accessed servers). Thus, the network latency is $\mathcal{O}(\tau)$ and the overlay hop is $\mathcal{O}(1)$. In Section 5.1.1, we provide two methods: Eigenvalue Decomposition relies on matrix multiplication, having $\mathcal{O}\left(d|\widetilde{\Phi}|\right)$ time complexity, while the time complexity of Gradient Descent relies on the iteration number ξ and is $\mathcal{O}\left(\xi d|\Phi|\right)$.

a3) Complexity of Forwarding Graph Construction (Section 5.1.2): It mainly contains two parts: HNSW graphs and DT graphs. The time complexity to construct HNSW graphs is $\mathcal{O}\left(d|\Phi|\log|\Phi|\right)$. As for DT graphs, if d is 2, the time complexity is $\mathcal{O}\left(|\Phi|\log|\Phi|\right)$ and is $\mathcal{O}\left(|\Phi|^{\lceil d/2\rceil}\right)$ if $d \geq 3$. The server responsible for constructing the forwarding graph needs to know the virtual coordinates of all the edge servers in the system, thereby requiring $\mathcal{O}(\tau)$ network latency, $\mathcal{O}(1)$ overlay hop, $\mathcal{O}\left(|\Phi|\right)$ packet sizes, and $\mathcal{O}\left(|\Phi|\right)$ accessed servers.

b1) Complexity of Subarea Partition (Section 5.2.2): Subarea Partition divides the n-dimensional feature space into many subareas. We use the *Grouping and Pruning* method [1] with the time complexity $\mathcal{O}\left(\lambda n|\mathcal{X}||\widetilde{\mathbf{C}}|\right)$ where $|\mathcal{X}|$ denotes the number of sampled feature vectors and λ denotes the number of iterations.

b2) Complexity of Subarea VCs (Section 5.2.3): The complexity is similar to that of LandmarkVC and NormalVC. It has the time complexity $\xi d|\widetilde{\mathbf{C}}||\mathbf{C}|$ to compute the virtual coordinates of all feature subareas since $|\widetilde{\mathbf{C}}| \ll |\mathbf{C}|$.

c) Complexity of Data-Edge Association (Section 5.3.1): Here, we mainly analyze the complexity of OTIR. In general, discrete optimal transport is solved through the Sinkhorn algorithm, with the upper-bound time complexity $\mathcal{O}(|\mathbf{C}||\Phi|/\varepsilon^2)$ to find an ε-approximate solution [2].

In summary, the total complexity of the offline phase is $\mathcal{O}(\lambda n|\mathcal{X}||\widetilde{\mathbf{C}}|+|\Phi|^{\lceil d/2\rceil}+|\mathbf{C}||\Phi|/\varepsilon^2)$. Since it requires the pairwise latency from the landmark servers to all edge servers, the packet size is $\mathcal{O}\left(|\Phi||\widetilde{\Phi}|\right)$.

B. Complexity Analysis for the Online Stage. In the original version of the paper, we have the complexity of some components at the online state (see details in Section 6), including M-Projection on Features, Index Range Checking, Feature Operations, Data Insert/Delete, and Data Query. Here, we supplement the complexity analysis about Dynamic Node Joining and Dynamic Node Departing.

i) Complexity of Dynamic Node Joining. For a newly joined edge server p, we need to compute its virtual coordinate $\boldsymbol{e}_p$, construct its forwarding table T_p, update the forwarding tables of its neighbor servers Φ_p existing in T_p, and update the index tables of both the edge server p and its neighbor servers Φ_p. The virtual coordinate $\boldsymbol{e}_p$ can be obtained by solving the NormalVC problem, which only requires the virtual coordinates of landmark servers and the latency vector from the server p to the landmark servers. It requires $\mathcal{O}\left(|\widetilde{\Phi}|\right)$ accessed servers and $\mathcal{O}\left(|\widetilde{\Phi}|\right)$ packet sizes. The time complexity is $\mathcal{O}(\xi d|\widetilde{\Phi}|)$. The forwarding table T_p can be obtained through the randomized incremental construction algorithm for DT graphs [3], with the time complexity $\mathcal{O}(d\log|\Phi|)$. Note that besides the server p, the forwarding tables of some relevant servers also need to be updated ($\mathcal{O}\left(|T_p|+\log|\Phi|\right)$). After updating the forwarding tables, each one of the neighbor servers Φ_p determines whether the kept index

Yuchen Sun (sunyuchen18@nudt.edu.cn), Lailong Luo (luolailong09@nudt. edu.cn), Deke Guo (dekeguo@nudt.edu.cn), and Bangbang Ren (renbangbang11@nudt.edu.cn) are with Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology (NUDT), Changsha, China. Li Liu (dreamliu2010@gmail.com) is with the College of Electronic Science, NUDT, Changsha, Hunan, China.
The corresponding authors are Lailong Luo and Deke Guo.

TABLE 1
Analysis for Time & Space Complexity and Network Costs

Component	Time Complexity	Space Complexity	Net Latency	Overlay Hops	Accessed Servers	Packet Size
LandmarkVC	Eig: $\mathcal{O}(\lvert\widetilde{\Phi}\rvert^2 \log \lvert\widetilde{\Phi}\rvert)$; Grad: $\mathcal{O}(\xi d\lvert\widetilde{\Phi}\rvert^2)$	$\mathcal{O}(\lvert\widetilde{\Phi}\rvert^2)$	$\mathcal{O}(\tau)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\widetilde{\Phi}\rvert)$	$\mathcal{O}(\lvert\widetilde{\Phi}\rvert^2)$
NormalVC	Eig: $\mathcal{O}(d\lvert\widetilde{\Phi}\rvert)$; Grad: $\mathcal{O}(\xi d\lvert\widetilde{\Phi}\rvert)$	$\mathcal{O}(\lvert\widetilde{\Phi}\rvert)$	$\mathcal{O}(\tau)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\widetilde{\Phi}\rvert)$	$\mathcal{O}(\lvert\widetilde{\Phi}\rvert)$
Forwarding Graph (Merge)	$\mathcal{O}(\lvert\Phi\rvert^{\lceil d/2\rceil})$, $d{\geq}3$; $\mathcal{O}(\lvert\Phi\rvert \log \lvert\Phi\rvert)$, $d{=}2$	$\mathcal{O}(\lvert\Phi\rvert^{\lceil d/2\rceil})$	$\mathcal{O}(\tau)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\Phi\rvert)$	$\mathcal{O}(\lvert\Phi\rvert)$
Subarea Partition	$\mathcal{O}(\lambda n\lvert\mathcal{X}\rvert\lvert\widetilde{\mathbf{C}}\rvert)$	$\mathcal{O}(d\lvert\widetilde{\mathbf{C}}\rvert)$	-	-	-	-
Subarea VCs (Grad)	$\mathcal{O}(\xi d\lvert\widetilde{\mathbf{C}}\rvert\lvert\mathbf{C}\rvert)$	$\mathcal{O}(\lvert\widetilde{\mathbf{C}}\rvert\lvert\mathbf{C}\rvert)$	-	-	-	-
OTIR (Sinkhorn)	$< \mathcal{O}(\lvert\mathbf{C}\rvert\lvert\Phi\rvert/\varepsilon^2)$	$\mathcal{O}(\lvert\mathbf{C}\rvert\lvert\Phi\rvert)$	-	-	-	-
Offline Total	$\mathcal{O}(\lambda n\lvert\mathcal{X}\rvert\lvert\widetilde{\mathbf{C}}\rvert{+}\lvert\Phi\rvert^{\lceil d/2\rceil}{+}\lvert\mathbf{C}\rvert\lvert\Phi\rvert/\varepsilon^2)$	$\mathcal{O}(\lvert\Phi\rvert^{\lceil d/2\rceil}{+}\lvert\widetilde{\mathbf{C}}\rvert\lvert\mathbf{C}\rvert)$	$\mathcal{O}(\tau)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\Phi\rvert)$	$\mathcal{O}(\lvert\Phi\rvert\lvert\widetilde{\Phi}\rvert)$
Dynamic Node Joining	$\mathcal{O}(n\lvert\mathcal{X}_p\rvert\lvert T_p\rvert + \xi d\lvert\widetilde{\Phi}\rvert + d\log\lvert\Phi\rvert)$	$\log\lvert\mathcal{X}_p\rvert\lvert\mathbf{C}\rvert$	$\mathcal{O}(\tau)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert T_p\rvert{+}\log\lvert\Phi\rvert)$	$\mathcal{O}(\lvert\mathcal{X}_p\rvert)$
Dynamic Node Departing	$\mathcal{O}(n\lvert\mathcal{X}_p\rvert\lvert T_p\rvert)$	$\log\lvert\mathcal{X}_p\rvert\lvert\mathbf{C}\rvert$	$\mathcal{O}(\tau)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert T_p\rvert{+}\log\lvert\Phi\rvert)$	$\mathcal{O}(\lvert\mathcal{X}_p\rvert{+}\lvert T_p\rvert^2)$
Data Insert/Delete (OTIR)	$\mathcal{O}(d\log\lvert\Phi\rvert{+}\max_{p\in\Phi_D} n\log\lvert\mathcal{X}_p\rvert\lvert\mathbf{C}\rvert)$	$\mathcal{O}(\max_p n\log\lvert\mathcal{X}_p\rvert\lvert\mathbf{C}\rvert)$	$\mathcal{O}(\tau)$	$\mathcal{O}(\log\lvert\Phi\rvert)$	$\mathcal{O}(R\log\lvert\Phi\rvert)$	$\mathcal{O}(R\log\lvert\Phi\rvert)$
Data Query (OTIR)	$\mathcal{O}(dM\log\lvert\Phi\rvert{+}\max_{p\in\Phi_D} n\log\lvert\mathcal{X}_p\rvert\lvert\mathbf{C}\rvert)$	$\mathcal{O}(\max_p n\log\lvert\mathcal{X}_p\rvert\lvert\mathbf{C}\rvert)$	$\mathcal{O}(\tau)$	$\mathcal{O}(\log\lvert\Phi\rvert)$	$< \mathcal{O}(M\log\lvert\Phi\rvert)$	$\mathcal{O}(M\log\lvert\Phi\rvert)$

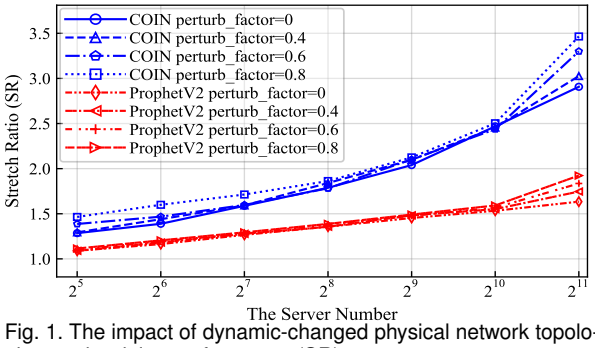

Fig. 1. The impact of dynamic-changed physical network topologies on the delay performance (SR).

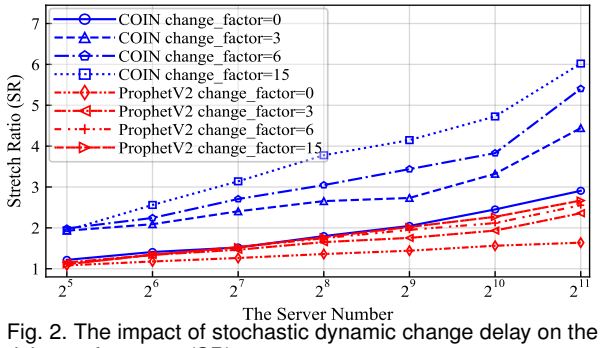

Fig. 2. The impact of stochastic dynamic change delay on the delay performance (SR).

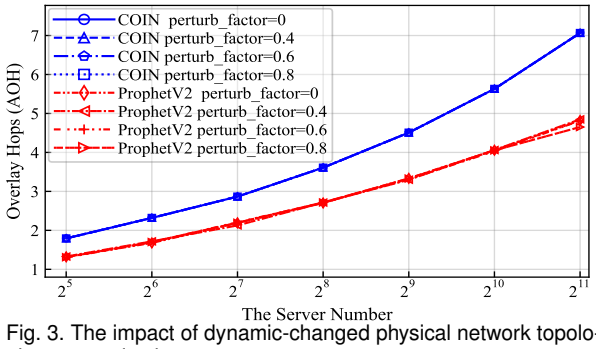

Fig. 3. The impact of dynamic-changed physical network topologies on overlay hops.

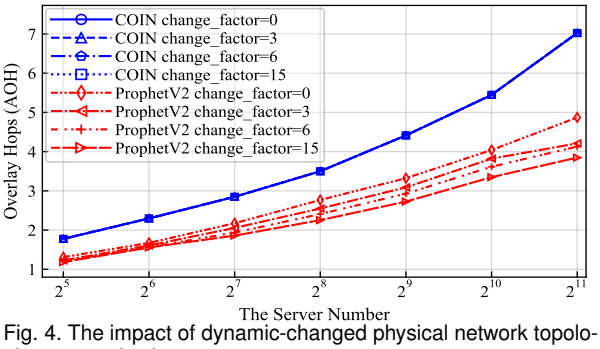

Fig. 4. The impact of dynamic-changed physical network topologies on overlay hops.

entries are still stored locally or are transferred to the new server p. It has the time complexity of $\mathcal{O}(n|\mathcal{X}_p||T_p|)$. We assume that the number of landmark servers is the logarithm of the number of edge servers in the system, *i.e.*, $\mathcal{O}\left(|\widetilde{\Phi}|\right) \sim \mathcal{O}\left(\log|\Phi|\right)$. Therefore, Dynamic Node Joining has $\mathcal{O}(n|\mathcal{X}_p||T_p| + \xi d|\widetilde{\Phi}| + d\log|\Phi|)$ time complexity, $\log|\mathcal{X}_p||\mathbf{C}|$ space complexity, $\mathcal{O}\left(|T_p|+\log|\Phi|\right)$ accessed servers and $\mathcal{O}(|\mathcal{X}_p|)$ packet sizes.

ii) Complexity of Dynamic Node Departing. The departing server p first hands over its index entries to its neighbor servers, that is, the server with the closest virtual coordinate to the index entry besides the server p itself. Then, we update the forwarding tables of the neighbor servers Φ_p by reconstructing a DT subgraph on the virtual coordinates of Φ_p. The time complexity is in index migration ($\mathcal{O}(n|\mathcal{X}_p||T_p|)$). For network costs, it requires $\mathcal{O}\left(|T_p|+\log|\Phi|\right)$ accessed servers and $\mathcal{O}\left(|T_p|+\log|\Phi|\right)$ packet sizes.

2 Impacts of Latency Measurement Errors

Since the edge servers in the proposed ProphetV2 use static virtual coordinates in the d-dim Euclidean virtual space, the measurement errors in the latency matrix do have impacts on the latency performance for data insert/query.

Firstly, we analyze the factors contributing to measurement errors in inter-node network latency, and summarize two factors: i) One is the dynamic change of real-time network loads, including random changes, sudden changes, or periodic changes; and ii) the other is changes in physical network topologies, such as node joining or departing, or changes in the link state between nodes.

Factor 1: Due to the dynamic changes in network load and network device status, the network latency between edge servers may have random changes, thereby introducing noise to the measured latency. This is a core cause of measurement errors. This factor does not cause changes in the physical network topology itself and therefore has little impact on the latency performance of ProphetV2. Here, we simulate this measurement error by adding random noise to the measured latency matrix. In detail, we multiply each element of the latency matrix θ_{ij} by a random number ϵ_{ij} to simulate the measurement error of the latency. Note that the random number ϵ_{ij} is sampled from the uniformly distributed $\mathcal{U}(1-0.5\times perturb\_factor, 1+0.5\times perturb\_factor)$. The latency matrix injected with noise is finally processed into a symmetric matrix.

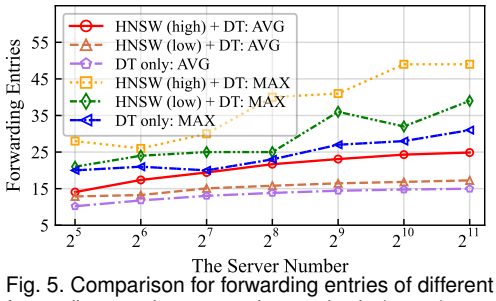

Fig. 5. Comparison for forwarding entries of different forwarding graph construction methods ($d = 3$).

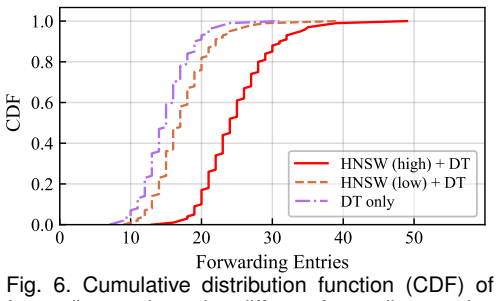

Fig. 6. Cumulative distribution function (CDF) of forwarding entries using different forwarding graphs.

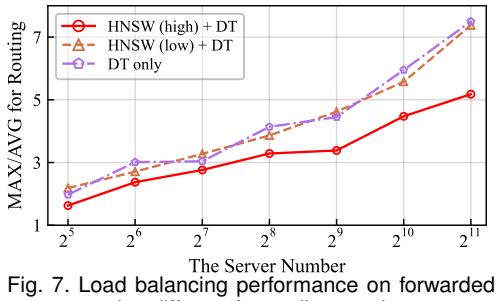

Fig. 7. Load balancing performance on forwarded requests using different forwarding graphs.

Factor 2: Changes in the physical network topology will introduce bias errors in the measurement of the latency matrix. Furthermore, with the continuous operation of the system, the current physical network topology will be significantly different from the network topology measuring the latency matrix, which leads to cumulative error in the latency measurement. Here, we simulate this measurement error by modifying the latency of some node pairs. In detail, we randomly select some node pairs and connect or disconnect their links. The number of disconnected or connected links are both $change\_factor \times N$ where N denotes the node number. We achieve this goal by directly modifying the latency matrix. If we want to disconnect the link between node i and node j, we set the corresponding value θ_{ij} in the latency matrix $\mathbf{\Theta}$ to 0. If we want to link two nodes i and j, we randomly set the corresponding value θ_{ij} to a small latency. The Floyd algorithm is used to compute the final latency matrix.

Next, we evaluate the impacts of these two factors on the latency performance of ProphetV2, via two metrics: Stretch Ratio (SR) and Average Overlay Hops (AOH). In summary, the experimental results show the robustness of our ProphetV2 against latency measurement errors. Fig. 1, Fig. 2, Fig. 3, and Fig. 4 compare the performance between COIN and ProphetV2 under different system scales. Note that the dimension of the virtual space is 3 (*i.e.*, $d = 3$). Fig. 1 and Fig. 3 show the impact of the factor 1 (*i.e.*, noise errors) on the latency performance. We can find that noise errors have minor harm on both Stretch Ratio and Average Overlay Hops. Fig. 2 and Fig. 4 show the impact of the factor 2 (*i.e.*, bias errors) on the latency performance. Although the bias errors have some performance impacts, it is still tolerable.

3 Impacts of Algorithms for Forwarding Graph Construction

We evaluate the metrics you mentioned through experiments using three methods for forwarding graph construction. Note that the dimension of the virtual space is 3 (*i.e.*, $d = 3$) in the experiments.

Firstly, we discuss the maximum degree of forwarding graphs. In the original HNSW algorithm, the search process is from top to bottom, layer by layer, and there is no relationship between layers. The maximum vertex degree of each layer in the original HNSW graph is $\mathcal{O}(1)$. In ProphetV2, we combine multiple layers into one and we can treat the number of layers as a constant. From this, we can get that the maximum vertex degree of the combined HNSW graph is also constant. And the maximum vertex degree of a DT graph is also constant if the dimension d is fixed. Therefore, all the three methods for forwarding graph construction, HNSW (high) + DT, HNSW (low) + DT, DT only, have $\mathcal{O}(1)$ maximum vertex degree in a fixed d. Fig. 5 shows the experiment results, that is, the average and maximum vertex degree using the three types of forwarding graphs. Fig. 6 shows the Cumulative Distribution Function (CDF) of forwarding entries using different forwarding graphs. Note that the server number is 2048. We can find that the maximum vertex degree is in the same order of magnitude as the average vertex degree.

Secondly, we evaluate the load-balancing performance on the number of forwarded requests. As shown in Fig. 7, the method "HNSW (high) + DT" performs better than the other two. Some edge servers do take on more request forwarding tasks, and the server heterogeneity on forwarding capabilities is also ignored. This will be considered in our future work.

References

[1] D. Baranchuk, A. Babenko, and Y. Malkov, "Revisiting the inverted indices for billion-scale approximate nearest neighbors," in *ECCV*, 2018.
[2] K. Pham, K. Le, N. Ho, T. Pham, and H. Bui, "On unbalanced optimal transport: An analysis of sinkhorn algorithm," in *International Conference on Machine Learning*. PMLR, 2020, pp. 7673–7682.
[3] L. J. Guibas, D. E. Knuth, and M. Sharir, "Randomized incremental construction of delaunay and voronoi diagrams," *Algorithmica*, 1992.