2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

Distributed Client-Server Optimization for SLAM with Limited On-Device Resources

Yetong Zhang[1,2], Ming Hsiao[1], Yipu Zhao[1], Jing Dong[1], and Jakob J. Engel[1]

***Abstract*— Simultaneous localization and mapping (SLAM) is a crucial functionality for exploration robots and virtual/augmented reality (VR/AR) devices. However, some of such devices with limited resources cannot afford the computational or memory cost to run full SLAM algorithms. We propose a general client-server SLAM optimization framework that achieves accurate real-time state estimation on the device with low requirements of on-board resources. The resource-limited device (the client) only works on a small part of the map, and the rest of the map is processed by the server. By sending the summarized information of the rest of map to the client, the on-device state estimation is more accurate. Further improvement of accuracy is achieved in the presence of on-device early loop closures, which enables reloading useful variables from the server to the client. Experimental results from both synthetic and real-world datasets demonstrate that the proposed optimization framework achieves accurate estimation in real-time with limited computation and memory budget of the device.**

I. Introduction

In the past decade, the applications of SLAM on light-weight devices have increased significantly, which include AR/VR headsets [20], small robots [10] and smart phones [23]. For such devices, the power consumption poses a strict bottleneck on the running time of general SLAM algorithms, which are usually expensive in computation and memory consumption. For future applications that require further scaling down of device size and power limit, such as coin-size drones [11], [12], [19] and AR glasses, the applicability of conventional SLAM is greatly limited. For example, a state-of-the-art SLAM system, ORB-SLAM [17], normally runs on a multi-core CPU, and occupies memory up to 200MB. Meanwhile, a light-weight device such as a commercially available nano drone operates on an embedded system with less than 1MB of RAM [11]. The gap between the on-device resource availability and the state-of-the-art SLAM requirement is huge.

To achieve low computation and memory consumption, a common solution is to adopt sliding window or filtering approaches in SLAM. Such solutions only maintain a short memory of historic information, while dropping or marginalizing out-of-window information continuously. Due to the early fixation of linearization points and the accumulating nature of linearization error, the estimation accuracy of such solutions is limited [7].

In pursuit of a resource-efficient approach that does not suffer from aggregated linearization errors, several client-server SLAM frameworks [26] [16] [21] have been proposed. In each of these works, a powerful server operates on all variables and measurements in the history (i.e., the full graph), and the resource-limited client works on a small fraction of variables and measurements (i.e., a local graph). Therefore, the device can achieve both computation and memory efficiency due to its small problem size, while it benefits from the more accurate estimation provided by the server. Nevertheless, we argue that there are two limitations within the existing client-server SLAM frameworks. Firstly, the device does not have any uncertainty information of the measurements outside its local graph, which loses the chances of improving the estimation results on the device with those information. Secondly, if an incoming measurement is associated with variables outside the local graph of the device (e.g. loop closure to an early estimation), the device cannot acquire such measurement until the server finishes optimization with the new variables. Therefore, the chance for the device to use the loop closure measurements to improve its estimation result is delayed.

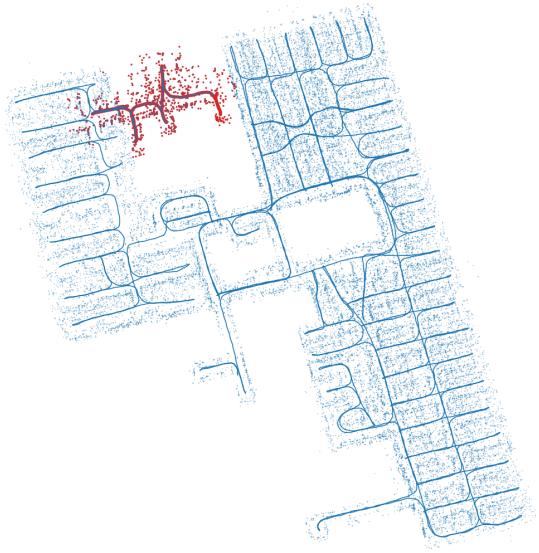

Fig. 1. An example of our proposed method running on a large indoor visual-inertial SLAM dataset. The entire area is around 50m×70m. The blue trajectory and map points are maintained on the server side, and the red ones are maintained on the device side. More details about this dataset are discussed in Section VI-C.

To solve resolve the aforementioned drawbacks, we propose a client-server SLAM optimization framework with the following improvements from other existing works. Firstly, the server regularly sends summarized information of the entire history including the uncertainties to the device, which

[1]Facebook Reality Labs Research, Redmond, USA {zhangyetong, mhsiao, yipuz, jingdong, jakob.engel}@fb.com
[2]College of Computing, Georgia Institute of Technology, Atlanta, USA yetong@gatech.edu

enhances the local graph that the on-device SLAM is working on. Secondly, the server is able to send back historic variables that are associated with the new variables on the device. As a result, the device can incorporate the measurements between the new and historic variables at an earlier time and perform an early loop closure thereafter. Thirdly, information sparsification is applied to the summarized information to bound the communication load. We evaluate our client-server SLAM optimization framework with both synthetic and real world datasets. The results show that our approach improves the real-time estimation results without increasing the computation load on the device.

II. Related Work

Client-server SLAM frameworks have been studied in several previous works. In [3], a group of robots is coordinated by a server to explore unknown environment. Each robot performs localization on its own using particle filter, while the server collects the maps from each robot and proposes points of interest to guide the robots. In [26], a graph-based approach was introduced to client-server SLAM framework. The server-side estimations are globally optimized, and shared with the client to rectify the client-side estimations. In [16] and [21], the server acts as a central hub of all the information from the clients, enabling the clients to load adjacent maps to improve their measurements. In [9], the robot offload heavy computation to the server, and use the latest available pose from the server to reset its estimations. However, none of these approaches incorporate the full information including uncertainties outside the local map in the device; nor did they make any systematic study on the communication timeline.

One inspiration of this work comes from the existing study on decentralized multi-robot SLAM problems [4], [14], where the information were summarized and shared between robots in the form of condensed measurements. When one robot is sending information to another robot, it identifies the shared variables in their own graphs, and marginalizes out all the rest of the variables to generate marginal factors. If a server can use such methods to summarize the information outside the local graph of the device and provide it to the device, it is possible to improve the on-device estimations.

Another inspiration of this work comes from [25]. Even though it operates on a single machine, it is similar to a client-server approach because the inference problem is split into a filter component with lower latency but lower accuracy, and a smoother component with higher accuracy but higher latency. The full joint density is factorized in a way such that the two components of the inference problem are joined by a group of *separator variables*, which is adopted in our client-server framework as well.

Information sparsification methods for SLAM problems have been well studied, and they can be applied in our framework to effectively bound the communication load. Two popular categories of information sparsification algorithms are generic linear constraints (GLC) [2], [24] and nonlinear factor recovery (NFR) [15]. GLC approaches uses a set of linear factors to approximate the joint probability of the original factor, while NFR uses the nonlinear factors chosen by the user. In [24], an efficient sparsification technique is proposed by ignoring the covariance between variables.

III. Problem Formulation

Before introducing the proposed methods, we first give a short introduction on 1) modeling a SLAM problem with a factor graph, and 2) how factor graphs are used in a client-server optimization setting as background.

A. Factor Graph for SLAM Inference

Factor graph [8] is a type of bipartite graph that consists of two types of nodes: variable nodes and factor nodes. Each variable node represents an unknown variable $\theta \in \Theta$, where Θ is the set of all unknown variables; each factor node represents the potential $P(Z_i|\Theta_i)$ induced by a measurement Z_i, where Θ_i represents all variables associated to the measurement. The entire factor graph represents a full joint probability $P(\Theta|Z)$ as Eq. 1. As an inference problem, our goal is to compute the probability distribution of variables Θ provided with all the sensor measurements Z, and the maximum a posterior (MAP) estimate is acquired by Eq. 2.

$$P(\Theta|Z) \propto \prod_i P(Z_i|\Theta_i) \tag{1}$$

$$\Theta^* = \arg\max_{\Theta} \prod_i P(Z_i|\Theta_i) \tag{2}$$

B. Client-Server Optimization

We formulate a client-server SLAM problem as a factor graph inference problem, in which the client and server have access to different parts of the factor graph (see Fig. 2). At any given time, we define the variables and measurements that only exist in the server as *historic variables* Θ_h and *historic measurements* Z_h, the variables and measurements that exist in both the server and the device as *separator variables* Θ_s and *separator measurements* Z_s, the variables that only exist in the device as *new variables* Θ_n. The measurements that associate with new variables are defined as *new measurements* Z_n. We further distinguish the new measurements that associate with some historic variables as *loop closure measurements* Z_{lc}, and define such historic variables as *loop closure variables* Θ_{lc}. We use Z_{ns}, Θ_{ho} to represent the rest of new measurements and the rest of historic variables, respectively.

Ideally the best possible estimation can be acquired by solving the full SLAM problem, which corresponds to the entire factor graph in Fig. 2.

$$\begin{aligned}\Theta^* = &\arg\max_{\Theta} P(Z_h|\Theta_h, \Theta_s)P(Z_s|\Theta_s) \\ &P(Z_{lc}|\Theta_n, \Theta_{lc})P(Z_{ns}|\Theta_n, \Theta_s)\end{aligned} \tag{3}$$

However, such solutions cannot be acquired in existing client-server frameworks. Instead, the on-device optimization problem is performed only on the part of the factor graph that exists in the device:

$$\{\Theta_n^*, \Theta_s^*\} = \arg\max_{\Theta_n, \Theta_s} P(Z_s|\Theta_s)P(Z_{ns}|\Theta_n, \Theta_s) \tag{4}$$

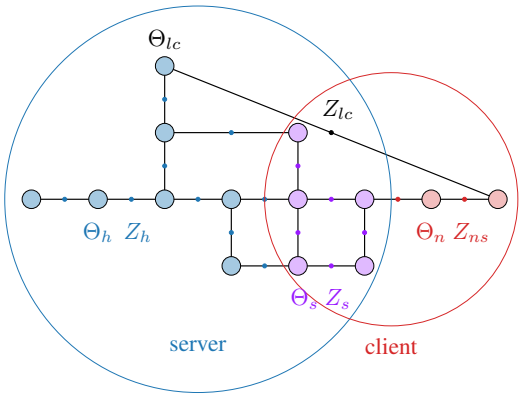

Fig. 2. The factor graph representation of a toy client-server SLAM optimization problem. The server optimizes on the factor graph within the blue region, and the device optimizes on the factor graph within the red region. The separator variables Θ_s and separator factors Z_s are marked in magenta, and they coexist in both the server and device. The factor marked in black is identified as a loop closure factor Z_{lc}, because it associate with a new variable Θ_n and a historic variable Θ_h.

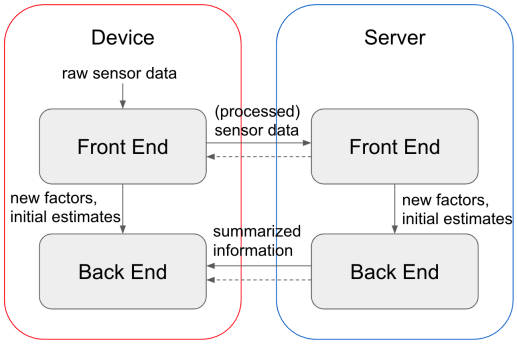

Fig. 3. Overall system architecture: raw data is acquired from the sensors on the device, and is passed to the two front-ends that are on the device and the server respectively. Both front-ends process sensor data to generate new factors and initial estimates of new variables for the back end. The back-end of the server regularly sends summarized information to the back-end of the device. The dashed arrows represent the server sending back loop-closure information to the device (Section IV-C), where the back-end of the server sends the prior on loop closure variables and the front-end sends the loop closure factors and features associated with the loop closure variables. In the spatial implementation in Section V-A, the dashed arrows additionally represent the server reloading "nearby" graph to the device.

Note that the problem in Eq. 4 differs from the problem in Eq. 3 by missing two terms: $P(Z_h|\Theta_h, \Theta_s)$, the full information of the factors and variables that only exist in the server, and $P(Z_{lc}|\Theta_n, \Theta_{lc})$, the loop closure measurements. Both types of the missing information are helpful in the estimation of the new states Θ_n.

The goal of our framework is to provide the real-time state estimation as close as possible to the full SLAM solution in Eq. 3 with the limited computation resources on the device. As a result, we will take both $P(Z_h|\Theta_h, \Theta_s)$ and $P(Z_{lc}|\Theta_n, \Theta_{lc})$ into account in the proposed framework.

IV. Method

A. System Architecture

In our framework, a client and a server run a SLAM algorithm together in a collaborative way. The goal of the client is to provide real-time estimates. The goal of the server is to provide accurate estimates, which can be used by the device to rectify its estimation results. The overall system structure is shown in Fig. 3.

Both the device and the server have a front-end and a back-end. Both front-ends operate in a similar way: they process the sensor data, create factors corresponding to the measurements derived from the sensor data, and provide initial estimates of the new variables associated with the measurements. However, the factors and the timing of generating them are different in these two front-ends. Since the client stores both the new and separator variables, the client front-end can add the new factors among them, $P(Z_{ns}|\Theta_n, \Theta_s)$, to the client factor graph immediately to provide fast real-time estimates. However, since only the server stores all variables, the loop closure measurements Z_{lc} that associated with variables outside the scope of the device can only be discovered in the front end of the server. Therefore, the server front-end adds loop closure factors, $P(Z_{lc}|\Theta_n, \Theta_{lc})$, in addition to the same new factors added in the client at a later time to provide slower but more accurate estimates.

Both back-ends maintain their own factor graphs that collect the factors generated by their corresponding front-ends, and compute the MAP estimates of the variables in their factor graphs. The server back-end keeps all the variables and factors, while old variables are removed from the device back-end to bound the memory size.

B. Summarized Information

In factor graphs, a common approach to preserve only part of the variables while keeping the rest information approximated is marginalization [22]. By applying marginalization on a factor graph, the full joint probability is factorized into the form of $P(\Theta_r, \Theta_e) = P(\Theta_r)P(\Theta_e|\Theta_r)$, where Θ_e represents the variables to be marginalized out, and Θ_r represents the remaining variables.

Fig. 4 illustrates the process of the server generating the summarized information, and the device incorporating the summarized information in its factor graph. To summarize the part of the factor graph that is unavailable in the device, we can apply Eq. 5 to marginalize out the historic variables Θ_h and get the remaining marginal term $P(\Theta_s|Z_h)$, which is correct up to the fixed linearization point. The linearization point is expected to be the best possible estimate at each time step, since the server performs global optimization on all the accumulated measurements. Therefore, we can pass the marginalization factor $P(\Theta_s|Z_h)$ to the device as a theoretically accurate approximation of the rest of the graph.

$$P(\Theta_s, \Theta_h|Z_h) = P(\Theta_s|Z_h)P(\Theta_h|\Theta_s, Z_h) \quad (5)$$

In practice, it is desirable to use an inference algorithm on the server that is efficient in both incremental optimization and generating summarized information, as the execution time of the two processes is directly related to the delay for the server to send out the summarized information. In our implementation, we use the iSAM2 algorithm [13] for the server in our implementation, as it excels in incremental

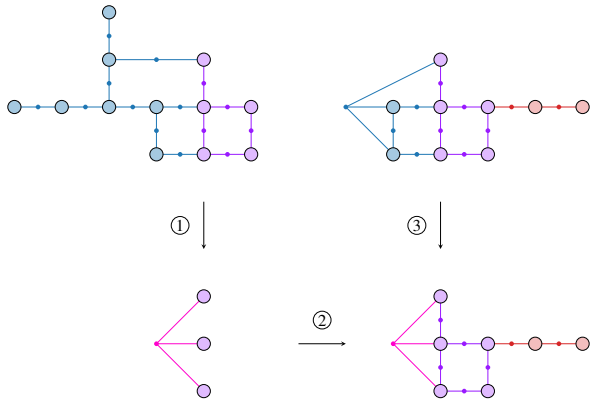

Fig. 4. The summarized information is generated by the server, and incorporated in the device. 1) The server marginalizes the history variables marked in blue, and generate marginal factors (marked in pink) on the separator variables. 2) The server sends the marginalization factor to the device. 3) The device remove old variables and factors marked in blue, and update the marginalization factor.

sparse nonlinear optimization by utilizing the Bayes tree structure. It is noted in [5] that the Bayes tree data structure also provides convenience in generating marginal factors.

C. Early Loop Closure

Through inspecting the communication schedule of a standard approach described in Section IV-B (Fig. 5), we discovered that there is an unavoidable delay t_d for the device to incorporate the loop-closure measurements Z_{lc} into optimization after the new variables Θ_n are firstly observed in the sensor. Since the device do not store historic variables Θ_h, it can only get loop-closure information from the server. The total delay is composed of 4 parts: $t_d = t_s + t_f + t_b + t_r$, where t_s is the time to send measurement information from device to server; t_f is the operation time in the server to generate factors in the front-end; t_b is the time to perform optimization and summarize information in the back-end; t_r is the time to send summarized information from the server to the device. Normally, the back-end optimization on the server is the most time consuming part in t_d due to the large factor graph size in the server. This operation is especially time consuming when large loop closure is encountered, because lots of variables are involved in the loop.

Therefore, we propose methods to reduce the delay so that the server can incorporate such measurements at an earlier time. We note in Fig. 5 that the loop closure factors are generated before the server starts optimization. Therefore, the server can send additional loop closure information to the device right after the loop closure factors are detected (see the red arrow in Fig. 5), reducing the delay to $t_s + t_f + t_r$ by skipping the server back-end optimization time t_b.

To make use of the loop closure factors, the device needs to know the priors on the loop closure variables Θ_{lc}. As the server already passes the marginal on the separator variables $P(\Theta_s|Z_h)$, we can further apply a factorization to Eq. 5 as

$$\begin{aligned} &P(\Theta_s, \Theta_h|Z_h) \\ =&P(\Theta_s|Z_h)P(\Theta_{lc}|\Theta_s, Z_h)P(\Theta_{ho}|\Theta_s, \Theta_{lc}, Z_h) \end{aligned} \quad (6)$$

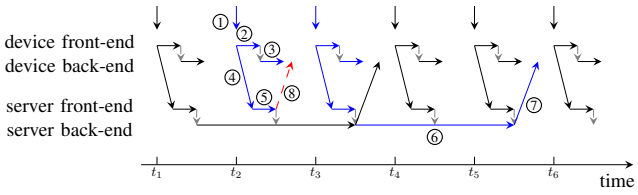

Fig. 5. A full cycle of updating estimates in our client-server system is marked in blue. 1) New sensor data is generated on the device at certain frequencies. 2) The device front-end processes the raw sensor data and generates factors Z_{ns}. 3) The device back-end performs a quick local optimization with the new measurements included. 4) At the same time, the device also sends the sensor data to the server. 5) The server front-end generates all factors including loop-closure factors ($Z_{ns} \cup Z_{lc}$). It will cache the factors until the server back-end is ready to add them to the factor graph. 6) The back-end of the server performs optimization on the entire factor graph with cached factors, and generate summarized information. 7) The server sends summarized information to the device. 8) The additional component for early loop closure method is marked in red. Upon detecting loop closure factors, the server sends loop closure information to the device.

Therefore, the posterior probability $P(\Theta_{lc}|\Theta_s, Z_h)$ can be sent to the device to incorporate prior information of loop closure variables Θ_{lc}, while maintaining consistency.

However, as the posterior $P(\Theta_{lc}|\Theta_s, Z_h)$ can be costly to compute, we approximate it with $P(\Theta_{lc}|Z_h)$, which is easy to compute. Notice that this approximation ignores the covariance between Θ_{lc} and Θ_s, which is similar to the information sparsification strategies used in [24]. Even though the approximation is not guaranteed to be probabilistic conservative, it will be overwritten by the theoretically correct summarized information from the server shortly after.

V. Implementation Details

A. Spatial vs. Temporal

The choice of separator variables can greatly affect the performance of the proposed algorithm. Since the front-end of the device can only discover the measurements associated with separator variables Z_{ns}, choosing better separator variables can enable the front-end to incorporate more measurements and generate more factors accordingly. Moreover, we would like to keep the separator variables that are more likely to have a larger update by the new measurements, so that we have a chance to relinearize the factors among them to provide better estimates.

There are two heuristics to select separator variables: the most recent ones (temporal), or the most adjacent ones (spatial). The implementation for temporal case is straightforward, since all variables are added in a temporal order. As for the spatial case, more work needs to be done. When the robot re-visits a previously observed region, some spatially adjacent variables may have been previously eliminated in the device, and becomes historic variables Θ_h (see Fig. 6). These nearby historic variables, as well as the measurements among them, need to be reloaded to the device.

In general, the spatial heuristic can generate more factors in the device than the temporal heuristic, since the new measurements are commonly related to the adjacent variables. Therefore, some loop closure measurements in the temporal case can be discovered in the front end with spatial heuristic.

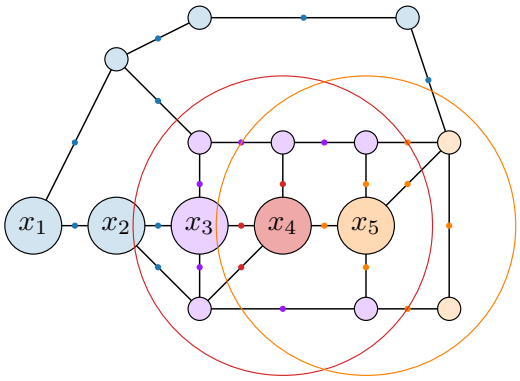

Fig. 6. An example demonstrating the spatial heuristic. Suppose the robot follows a trajectory x_1 - x_5 through a previously visited area. When the robot is at x_4, the adjacent region is marked by the red circle, and the separator variables are marked in magenta. When the robot moves to x_5, the variables and factors that become adjacent to the robot are marked in orange. Notice that such historic variables as well as the factors among them need to be reloaded to the device from the server.

However, the spatial heuristic may suffer from drifting, and it is more sensitive to communication delay than the temporal heuristic. In the case of severe drifting, the estimation of current pose drifts from the actual pose. Therefore, the adjacent variables identified by the current estimates may not be adjacent in the real world. In the case of large communication delay, the adjacent region identified by the server may no longer be adjacent to the device by the time the device receives the information, as the device may have moved out of the region during the delay t_d.

B. Information Sparsification

In Sec. IV-B, since the marginal factor is usually dense, its size, as in the form of an information matrix, is proportional to n^2, with n representing the number of variables it connects to. Therefore, when we send the marginal factors to the device, the message size may be too large, which increases the computation and memory costs in the device, and can potentially cause information congestion.

Therefore, we adopt information sparsification methods on the marginal factors, which generates a set of small sized factors to approximate the probability density induced by the marginal factors. We desire the algorithm to be simple, because the time for sparsification is part of t_b (in addition to the optimization and marginalization time), which contributes to the delay t_d. The algorithms that compare the KL-divergence for each pair of variables, such as Chow-Liu tree [1] is too computationally expensive for our application. As a result, we adopt the efficient Global Priors [24] sparsification approach in our implementation. We also show in the experiments that even assuming the same delay t_d, Global Priors generates similar results as the Chow-Liu tree sparsification methods in our applications.

VI. Experiments

A. Benchmarking setup

We implement the proposed algorithm based on GTSAM [6], which is a general C++ factor graph optimization library with an iSAM2 implementation. Both synthetic and real-world datasets are used to evaluate the performance of the proposed framework. The benchmarking is conducted in a simulation environment, where two threads are allocated to simulate server and client processes respectively. The server-client communication channel is simulated by passing message between the two threads. All evaluations are performed on a laptop computer.

We compare various configurations of our proposed method with a baseline, which simply uses the latest pose sent from the server to reset its estimations as in [9]. The six configurations include temporal separators, spatial separators, temporal separators with sparsification (marked as + S), temporal separators with early loop closure (marked as + LC), and temporal/spatial separators with both sparsification and early loop closure (marked as + S + LC). The pseudo ground-truth we adopt to evaluate accuracy is the batch optimization results at each iteration, since they are the best possible estimates with *all* the available information to date as in Eq. 3.

For all configurations, We show four major metrics for evaluation: average (per frame) translation error and rotation error for the real-time state estimation results on the device compared to pseudo ground-truth, average server processing time (t_b), and the average message size sent from server to device (amount of floating point numbers). Notice that we do not show the comparison of device side optimization time since the device in each experiment has very similar graph and variable size that result in very close time consumption. The additional loop closure factors have little influence on the graph size of the device because the number of loop closure variables Θ_{lc} is much fewer than the number of separator variables Θ_s in most cases, and Θ_{lc} can be eliminated in advance in the device, since they only connect to global priors and loop closure factors.

B. Synthetic 2D pose graph dataset

The first dataset we use is the M3500 synthetic dataset introduced in [18], which is a 2D pose graph dataset consisting of 3500 poses and 5600 relative pose constraints. we assume the incoming measurements emerge at a constant rate of $20ms$, with 10 new states at each step. We additionally fix the communication lag to $10ms$, and the size of separator variables to 300 poses.

From the benchmark results in Table I, we see that all configurations perform better than the baseline in terms of estimation accuracy. Since new states are most likely to associate with adjacent variables, the spatial configuration can incorporate more measurements and result in better estimation accuracy than the temporal configuration. Moreover, since the marginal factors in the spatial case are not as densely connected as in the temporal case, its message size is smaller. We further observe that information sparsification results in much smaller message sizes.

Furthermore, we study how the size of separator variables influences the estimation accuracy (see Fig. 7). Since larger separator variables size means more information is sent to the device, the estimation accuracy should be improved given

TABLE I

COMPARISON OF CLIENT-SERVER ALGORITHMS ON 2D SYNTHETIC POSE GRAPH DATASET

Algorithm	Translation error (m)	Rotation error (rad)	Server time (s)	Message (#float)
Baseline	0.407	0.0343	0.086	900
Temporal	0.357	0.0311	0.082	18725
Spatial	0.229	0.0196	0.082	5226
Temporal + S	0.331	0.0286	0.089	1336
Temporal + LC	0.235	0.0218	0.083	18834
Temporal + S + LC	0.192	0.0191	0.088	1451
Spatial + S + LC	0.191	0.0192	0.081	3280

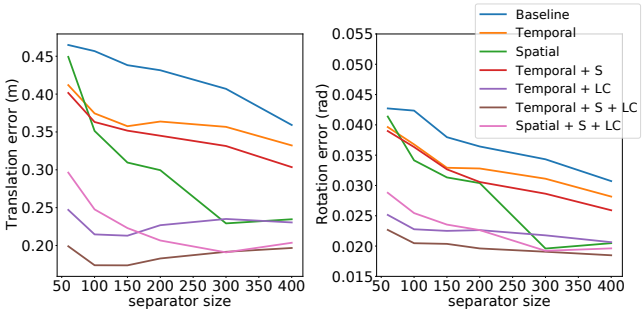

Fig. 7. Plot of the client estimation accuracy vs. the size of separator variables $|\Theta_s|$ for M3500 dataset.

larger separator variables size. The trend is clearly shown in the rotational and translational error plot. The performance with early loop closure helps a lot when separator size is small since it gains additional loop closure information for real-time state estimations. As the separator size increases, more loop closure measurements are likely to be incorporated in the device, and the effect of early loop closure is smaller. Therefore, the early loop closure strategy is most helpful when the graph size in the device is extremely limited.

It is interesting to discover that on this dataset, sparsification does not hurt the on-device estimation accuracy, and it even decreases the estimation error when early loop closure is applied, as "Temporal+S+LC" consistently outperforms "Temporal+LC" in Fig. 7. One possible explanation is that the adopted sparsification method [24] uses the same approximation strategy as the one used to generate priors for loop closure variables Θ_{lc}, which simply ignores the covariance between variables. Since the approximation strategy may not be conservative, it is possible that the non-conservative approximations of the priors on the separator variables Θ_s and loop closure variables Θ_{lc} are "balanced", therefore resulting in better estimates.

It is worth noting that the message size is only computed for the back-end communication. The communication in front-ends depends on the specific implementation that addresses data association.

C. Real-world visual-inertial SLAM dataset

The second dataset we use to evaluate our approach is a visual-inertial SLAM dataset, which is collected internally at Facebook. The sequence is collected by a custom rig, where multiple cameras and an IMU are mounted on a headset rigidly. The size of the mapped area is about 50m by 70m,

TABLE II

COMPARISON OF CLIENT-SERVER ALGORITHMS ON REAL-WORLD VISUAL-INERTIAL DATASET

Algorithm	Translation error (m)	Rotation error (rad)	Server time (s)	Message (#float)
Baseline	0.165	0.0172	1.29	663
Temporal	0.142	0.0157	1.40	64083
Spatial	–	–	–	–
Temporal + S	0.169	0.0171	1.46	1820
Temporal + LC	0.129	0.0139	1.38	65220
Temporal + S + LC	0.129	0.0140	1.45	2237
Spatial + S + LC	–	–	–	–

and the raw sensor stream sequence is about 30 minutes long. We get visual landmark information and pose-to-pose constraints from an in-house key-frame based front-end. In Fig. 1 we show the server and device map built by our algorithm (temporal separators configuration) close to the end of the sequence. The new measurements are sent to the server and the device at approximately 0.3s per 10 key frames, and we set the communication delay to 0.1s.

Since we have much denser graph in this visual-inertial SLAM dataset than the previous pose graph dataset, the device can only keep a smaller number of separator variables, which is set as 200 in all configurations.

From the results in Table II, we can again observe that incorporating the marginal uncertainty of temporal separators outperforms the baseline in accuracy. Moreover, temporal with sparsification and early loop closure increases estimation accuracy over the baseline without severely increasing the message size. The advantages of early loop closure would not have been acquired properly if we do not have uncertainty information over the separator variables.

However, both spatial cases fail due to communication delay. By the time the device receives the summarized information form the server, the current location of the device has moved out of the "adjacent region" marked by the separator variables. Therefore, the device fails to connect the new variables with the separator variables, and cannot provide valid estimates. In practice, the device can reject the summarized information from the server, and operate as a fixed-lag smoother in such circumstances.

VII. CONCLUSION

In this paper, we propose a general client-server optimization framework to solve SLAM problems with limited on-device resources. The summarized information is sent from the server to the device to inform the device of the uncertainty information outside of its local map. By carefully inspecting the communication timeline, we propose methods to achieve on-device early loop closure. We further perform a systematic study on the temporal and spatial heuristics in selecting separator variables, and the influence of applying sparsification on the summarized information. Future works include applying the proposed framework in a real-world system, refining the framework to consider on-device power efficiency, and generalizing the server to work with multiple devices simultaneously.

REFERENCES

[1] N. Carlevaris-Bianco and R. M. Eustice. Generic factor-based node marginalization and edge sparsification for pose-graph SLAM. In *2013 IEEE International Conference on Robotics and Automation*, pages 5748–5755, 2013.

[2] Nicholas Carlevaris-Bianco and Ryan M Eustice. Long-term simultaneous localization and mapping with generic linear constraint node removal. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1034–1041. IEEE, 2013.

[3] D. A. Castro, C. F. Morales, and R. F. De la Rosa. Multi-robot SLAM on client-server architecture. In *2012 Brazilian Robotics Symposium and Latin American Robotics Symposium*, pages 196–201, 2012.

[4] A. Cunningham, V. Indelman, and F. Dellaert. DDF-SAM 2.0: Consistent distributed smoothing and mapping. In *2013 IEEE International Conference on Robotics and Automation*, pages 5220–5227, May 2013.

[5] A. Cunningham, M. Paluri, and F. Dellaert. DDF-SAM: Fully distributed SLAM using constrained factor graphs. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3025–3030, 2010.

[6] Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology, 2012.

[7] Frank Dellaert and Michael Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, 25(12):1181–1203, 2006.

[8] Frank Dellaert and Michael Kaess. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, 6(1-2):1–139, 2017.

[9] D. Fourie, S. Claassens, S. Pillai, R. Mata, and J. Leonard. Slamindb: Centralized graph databases for mobile robotics. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6331–6337, 2017.

[10] S. García, M. E. López, R. Barea, L. M. Bergasa, A. Gómez, and E. J. Molinos. Indoor SLAM for micro aerial vehicles control using monocular camera and sensor fusion. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 205–210, 2016.

[11] Wojciech Giernacki, Mateusz Skwierczyński, Wojciech Witwicki, Paweł Wroński, and Piotr Kozierski. Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering. In *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 37–42. IEEE, 2017.

[12] M. Hassanalian and A. Abdelkefi. Classifications, applications, and design challenges of drones: A review. *Progress in Aerospace Sciences*, 91:99 – 131, 2017.

[13] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In *2011 IEEE International Conference on Robotics and Automation*, pages 3281–3288, 2011.

[14] M. T. Lázaro, L. M. Paz, P. Piniés, J. A. Castellanos, and G. Grisetti. Multi-robot SLAM using condensed measurements. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1069–1076, 2013.

[15] Mladen Mazuran, Wolfram Burgard, and Gian Diego Tipaldi. Nonlinear factor recovery for long-term SLAM. *The International Journal of Robotics Research*, 35(1-3):50–72, 2016.

[16] John G. Morrison, Dorian Gálvez-López, and Gabe Sibley. MOARSLAM: Multiple operator augmented RSLAM. In Nak-Young Chong and Young-Jo Cho, editors, *Distributed Autonomous Robotic Systems*, pages 119–132, Tokyo, 2016. Springer Japan.

[17] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[18] Edwin Olson, John Leonard, and Seth Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2262–2269. IEEE, 2006.

[19] Matthew Piccoli and Mark Yim. Piccolissimo: The smallest micro aerial vehicle. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3328–3333. IEEE, 2017.

[20] G. Reitmayr, T. Langlotz, D. Wagner, A. Mulloni, G. Schall, D. Schmalstieg, and Q. Pan. Simultaneous localization and mapping for augmented reality. In *2010 International Symposium on Ubiquitous Virtual Reality*, pages 5–8, 2010.

[21] Patrik Schmuck and Margarita Chli. CCM-SLAM: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams. *Journal of Field Robotics*, 36(4):763–781, 2019.

[22] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[23] Jonathan Ventura, Clemens Arth, Gerhard Reitmayr, and Dieter Schmalstieg. Global localization from monocular SLAM on a mobile phone. *IEEE Transactions on Visualization and Computer Graphics*, 20(4):531–539, apr 2014.

[24] D. Wilbers, L. Rumberg, and C. Stachniss. Approximating marginalization with sparse global priors for sliding window SLAM-graphs. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 25–31, Feb 2019.

[25] Stephen Williams, Vadim Indelman, Michael Kaess, Richard Roberts, John J. Leonard, and Frank Dellaert. Concurrent filtering and smoothing: A parallel architecture for real-time navigation and full smoothing. *The International Journal of Robotics Research*, 33(12):1544–1568, 2014.

[26] He Zhang, Zifeng Hou, Nanjun Li, and Shuang Song. A graph-based hierarchical SLAM framework for large-scale mapping. In Chun-Yi Su, Subhash Rakheja, and Honghai Liu, editors, *Intelligent Robotics and Applications*, pages 439–448, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.