学術交流支援資金報告書
アカデミックプロジェクト ノーベル・コンピューティング：AIと脳科学
研究課題名：データ駆動ロボティクスの創出

川島英之 *

March 28, 2022

1 Motivation: TF

The TF library on ROS centrally manages the transformations between each coordinate system as a directed tree structure, allowing for efficient registration of transformation information between coordinate systems and computation of transformations between Coordinate systems [2]. Left side of Fig 1 shows a robot and two objects in a room. The tree on the right side represents the positional relationship between each coordinate system on the robot and objects. A node represents each coordinate system, and an edge indicates the existence of coordinate transformation information (CTI) from a child node to its parent node.

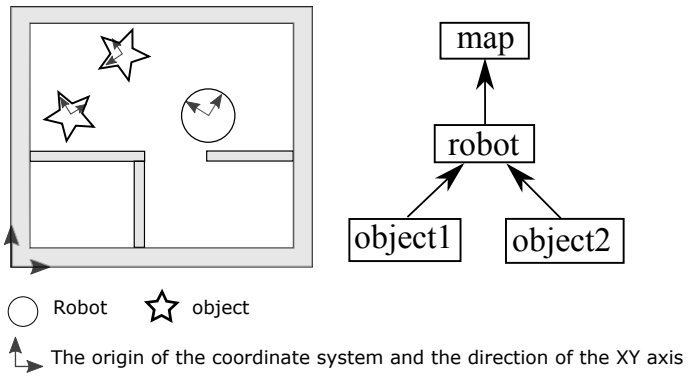

Figure 1: Robots in the room, and its TF tree structure.

*慶應義塾大学環境情報学部

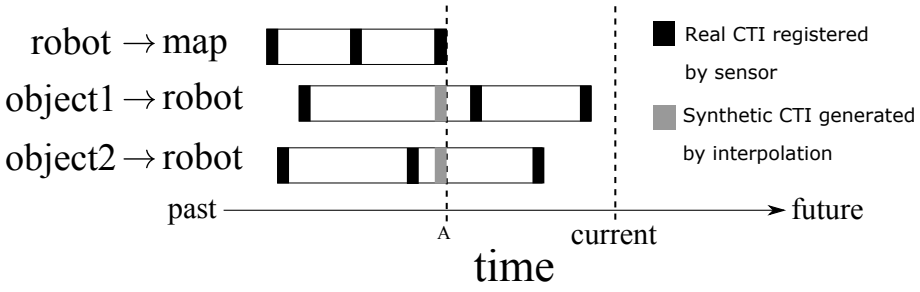

Figure 2: Timeline of location registration in 1

The CTI between frames can be registered at different times because the registration sources are sensor and sensors may have different periods. To provide a temporally consistent view of data, TF stores the CTI between each frame for 10 seconds by using linear interpolation. Fig 2 represents the timing at which the CTI between each frame is registered for the case shown in Fig 1.

In the situation of Fig. 1 and 2, the latest position relation from `object1` to `map` is calculated by TF as follows. First, TF checks each edge from `object1` to `map`. Next, TF checks the time at which we can provide the most recent and temporally consistent coordinate transformation possible for any edge. When the CTI of a certain time is stored or can be obtained by interpolation, TF considers that the CTI at that time can be provided. Fig 2 indicate that, the oldest time when the latest CTI is registered in `object1`→`robot` and `robot`→`map` is `robot`→`map`. Because the time A is the minimum among the maximum of two edges, time A is selected. Finally, TF computes the data for each edge at time A by interpolation, and conducts transformation by multiplying them.

2 Problem

TF manages positional relationships between coordinate systems, it has the following problems.

Problem 1: Giant Lock. The TF tree structure has an algorithm that prevents other threads from accessing the tree structure when one thread is accessing it. This may cause problems in throughput and latency in today's world where multi-core computing is the norm.

Problem 2: Data Freshness. Since the TF interface does not use the latest CTI as in the `object1` → `robot` example above, the data freshness is usually lost, which may deteriorate the quality in control or self-positioning. Besides, this specification requires frequent data registration even when the positional relationship between coordinate systems does not change much, resulting in unnecessary processing.

Problem 3: Transactional Correctness. As a solution to problem 2, it is not sufficient to provide an interface that performs inter-frame coordinate transformation calculations based on the latest coordinate transformation

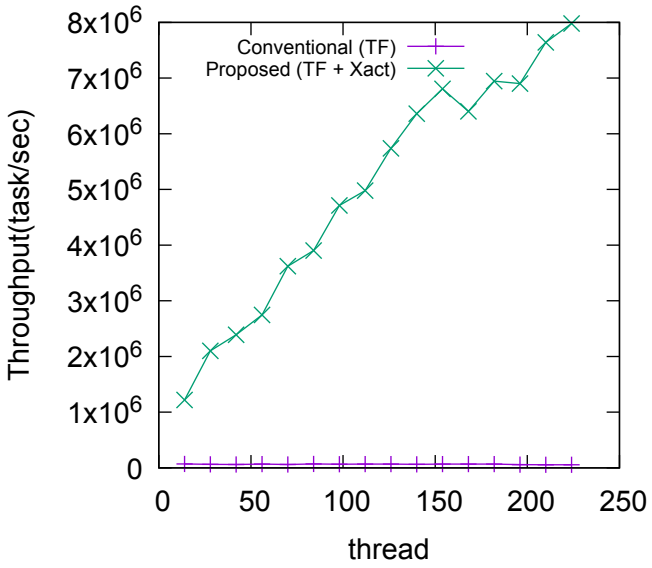

Figure 3: Throughput. Higher is better.

data. This is because the data consistency may be lost if inter-frame coordinate transformation calculations are performed based on the latest coordinate transformation data while multiple coordinate transformation data are being registered. It may produce transaction anomaly [1], which must be avoided.

3 Proposal

To solve problem 1, we present a decentralized design of TF with the fine-grained locking. We provide a mutex for a frame so that multile worker threads can access multiple frames in parallel when they invoke the conventional **lookupTransform()** and **setTransform()**. We implemented our locking system based on CCBench [3] so that it does not incur any contentions on read-read conflicts between two transactions.

To solve problem 2, we present new interfaces. **lookupLatestTransform()** returns the last data. The new interface may read stale data on object X while fresh data on object Y concurrently, which is temporally inconsistent referred to as anomaly [1]. To avoid the anomaly (i.e. problem 3), we present *transactional interfaces for TF* based on the two phase locking protocol. Our **lookupLatestTransformXact()** atomically retrieve the latest data of multiple coordinate transformations, and our **setTransformsXact()** atomically update the latest data of multiple coordinate transformations.

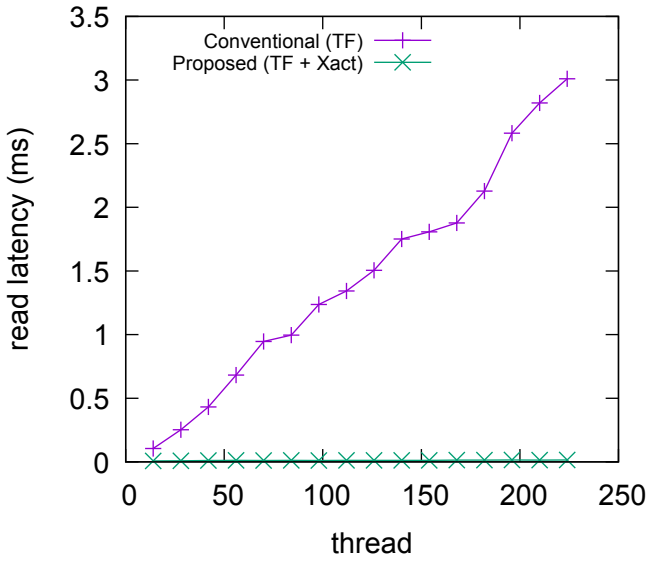

Figure 4: Latency. Shorter is faster.

4 Result

We installed ROS Melodic Morenia onto a Ubuntu 18.04 node, and experimented with a modified implementation of melodic-devel from the Github repository of the TF. Our code is available online [4].

We used a server with four Intel(R) Xeon(R) Platinum 8176 CPUs @ 2.10GHz. The total number of logical cores was 224. The workload is similar to the YCSB-A [3], and each transaction include 50% of read and write operations respectively. For all experiments, we chose a run time of 60 seconds, a time that yielded stable results.

The result of experiments are illustrated in Fig. 3, 4, and 5. As seen in Fig. 3, proposal shows scalable throughput. It is 143 times larger than that of TF on 224 threads. As seen in Fig. 4, proposal shows low latency even with 224 threads. It is 208 times shorter than that of TF. As seen in Fig. 5, the proposal shows low delay which is the difference from the timestamp of read access and the timestamp of data generation. On 224 threads, the delay of proposed TF is 132 times shorter than that of TF. These imporovement would be provided by fine grained locking because oritinal TF requires threads to acquire a giant lock even for accessing different frames.

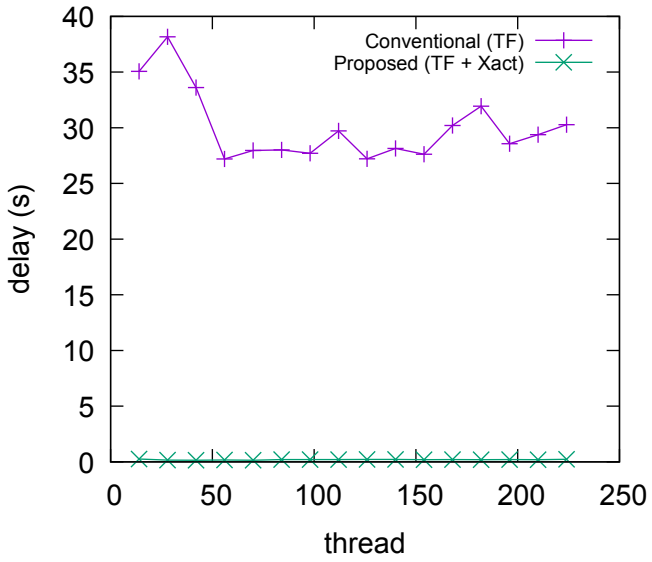

Figure 5: Delay. Shorter is more fresh.

5 Conclusion

TF tree access is not scalable due to a giant lock and does not provide the latest data. Our proposed method solved the problems by applying the fine-grained locking and 2 phase locking protocol. We showed that the proposed method achieved up to 143 times faster throughput, up to 208 times shorter latency, and up to 132 times data freshness than the conventional TF. Our code is available online [4].

References

[1] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ansi sql isolation levels. In *SIGMOD Conf.*, pages 1–10, 1995.

[2] T. Foote. tf: The transform library. In *TePRA*, pages 1–6, 2013.

[3] T. Tanabe, T. Hoshino, H. Kawashima, and O. Tatebe. An analysis of concurrency control protocols for in-memory databases with ccbench. *PVLDB*, 13(13):3531–3544, 2020.

[4] `https://github.com/Ogiwara-CostlierRain464/geometry2`, 2022. [Online; accessed 17-January-2022].