IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON INNOVATION AND APPLICATION OF INTELLIGENT PROCESSING OF DATA, INFORMATION AND KNOWLEDGE AS RESOURCES IN EDGE COMPUTING

Received October 1, 2019, accepted October 7, 2019, date of publication October 17, 2019, date of current version October 30, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2948021

Themis: An AST-Based Lock-Free Routes Synchronizing and Sharing System for Self-Driving in Edge Computing Environments

TE JIANG, TUN LU, AND NING GU
[1]School of Computer Science, Fudan University, Shanghai 201203, China
[2]Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China
[3]Shanghai Institute of Intelligent Electronics & Systems, Shanghai 200433, China

Corresponding author: Tun Lu (lutun@fudan.edu.cn)

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant 61932007 and Grant U1630115.

ABSTRACT The rapid development of self-driving technology has made self-driving cars come into reality, and some people have already adopted different levels of self-driving technology in real driving practices. With the help of self-driving technology, nowadays people can share and schedule their routes together while driving. However, currently, there are poor supports for such activities, as it demands highly responsiveness, strong consistency guarantees, and low transmission costs. As to support the novel scenario, we developed an edge computing oriented and highly efficient AST-Based lock-free synchronizing and sharing system called Themis on a fundamental itinerary planning model. We optimized the system by adopting partial replication, snapshots of history operations, and compression on consecutive operations strategies, which leave certain calculations at the cloud side, and reduce the amount and size of transmission data in the network. Besides, we have analyzed and proved the correctness of our scheme in detail; examining the significant improvements in performances through experiments.

INDEX TERMS Edge computing, collaborative tools, collaborative software, distributed computing.

I. INTRODUCTION

Nowadays, self-driving technology has been successfully applied to daily uses. More and more traditional car manufacturers and IT companies have joined the team to develop self-driving cars [16], many of which have already been put into use. In Schoettle et. al's earlier study [1] in 2014, though people still held serious concerns on safety issues in self-driving technology, they still had high expectations on it. While nowadays, such worries have been apparently alleviated with the growth of people's perception of self-driving, and some people have already accepted and adopted different levels of self-driving technologies in real daily driving practices [15], [17], [18].

It's common that in self-driving tours, people wanted to share their driving routes with their friends or families, for better scheduling and sharing of real-time locations and information, adjusting their routes for better travel experience or less time cost. And such needs would become especially urgent when driving in unfamiliar places with complicated road conditions or in disaster-stricken areas. However, real-time scheduling is not likely to be supported before, as the drivers cannot distract too much attention on scheduling. But now, with the development of self-driving, speech recognition and other technologies, people could gradually free their hands from holding the steering wheel and transform from hands input to other input methods; scheduling while driving now seems to be possible for drivers. Besides, nowadays many people voluntarily share real-time locations for predicting and reducing traffic jams. And we could also expect that, with sound safety and privacy-protect mechanisms, people are pleased to share their real-time driving routes for more accurate traffic jam prediction and better self-driving experiences in the near future.

However, the positions of the vehicles change quickly on maps as the cars move at high speeds, and the multi-user collaborating scheduling behavior would bring in-consistency problems. To share the real-time locations of all vehicles, the synchronizations under the fast-changing scenario would pose great pressure to the current network, and a strong consistency scheme is required to support multi-user scheduling behavior.

The associate editor coordinating the review of this manuscript and approving it for publication was Honghao Gao.

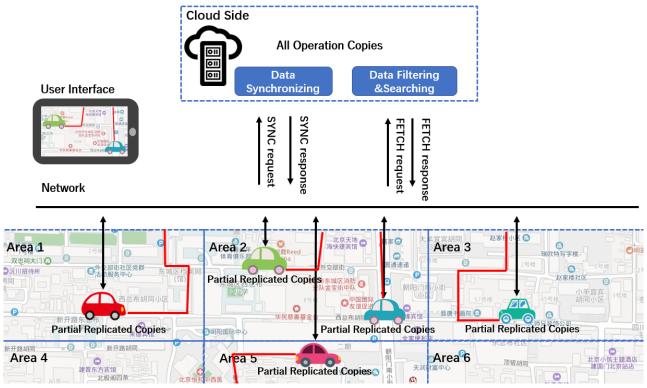

FIGURE 1. Data synchronization under partial replication architecture in self-driving scenario.

Adding locks is a straightforward method to guarantee consistency, and such strategy has been widely used in database systems [33], [35], and distributed applications [34]. However, in real-time interaction systems, like video game sharing [36], and collaborative editing [20], [21] scenarios, the responsiveness of the application is highly required; a delay time of more than 150ms could already affect user experience [37]. In the lock-based methods, operations on the same object could only be executed when it is unlocked or wait until the lock is released. Besides, the lock might always be held due to the loss of release signals in terrible network conditions, which is apparently not acceptable for real-time interaction systems.

Optimistic collaborative editing technique is rather a good choice to address the problem, as it provides real-time responsiveness and hides network delays, achieving final consistency. The collaborative editing techniques have been studied since the 1980s, firstly applied to the collaborative editing on documents [20]. And with the maturity of the basic algorithms, researchers extended to make optimization of the algorithms [31], [32] and tried to apply collaborative editing techniques to more editing scenarios [2], [24], [25].

Operation transformation [20], [23] and address space transformation [21] are the most representative optimistic collaborative editing algorithms, which could support real-time local responsiveness, and provide consistency maintenance, which well match the needs of our scenario. Besides, researchers also tried to apply a partial replication on the mobile commenting scenario by adopting AST algorithms [27], which inspired us to apply a partial replication strategy to our scenario to reduce transmission costs and improve responsiveness.

In this paper, we proposed and implemented an AST-based (Address Space Transformation [21]) lock-free and edge computing oriented highly efficient routes sharing and synchronizing technical system Themis on the basis of a fundamental real-time routes planning model [2], by applying the partial replication scheme and other optimizing strategies; making significant technical improvements on performances. The technical partial replicated architecture is shown in Figure 1.

The rest of the paper is organized as follows. Section 2 discusses the related work in achieving our efficient lock-free routes sharing and synchronizing model. The clear definition of the scenario, technical challenges, and basic model definitions are given in Section 3. And we presented our detailed technical scheme in Section 4, and analyzed and proved its correctness in Section 5. The experiments and performances are presented in Section 6. Section 7 concludes the paper by summarizing our contributions, limitations, and future research directions.

II. RELATED WORK

As our work is dedicated to supporting real-time itinerary planning and synchronizing under the self-driving scenario. In this section, we would discuss the traditional itinerary planning pattern and methods, and its possible shortages; the self-driving technology and its development; and work about lock-free real-time collaboration and consistency maintenance.

A. ITINERARY PLANNING PROBLEM

Itinerary planning was a classical research question, researchers have been studying the problem for decades. In the early days, itinerary planning problem was typically modeled as an orienteering problem(OP) or traveling salesman problem(TSP) [28]; researchers built their models by constructing basic elements and abstracting selected real conditions, like POI(Place of Interest), routes, and transports; meanwhile adding constraints, like time, budget, and weights. On the basis of the constrained model, researchers applied various methods to calculate the optimal or preferred values.

Search and greedy algorithms were the most common solutions [3], [4]; and also some heuristic methods [5], [7], [9], [30]. Besides, some researchers enriched the planning models by adding more elements and constraints, like time dependency [6], team planning needs [8]. By adopting these methods, possible routes could be automatically constructed in a really short time. However, such theoretical-optimal routes might not be able to cover all the essential variants and constraints, that the constructed scheme might not really fit users' interests. Besides, nowadays people pay less attention to time or budgets, that they prefer more personalized routes that perfectly match their own interests, which is hard to be achieved by adopting a model for general uses.

With the development of the mobile Internet industry, a large amount of traveler generated information is accumulated, like GPS tracks, pictures, user comments, and some other information on personal preferences. Some researchers tried to explore the value of these data and other rich travel-related information to make routes recommendations. They adopted collaborative filtering, spatial clustering,and other mature recommendation algorithms [10], [11] to make better personalized routes recommendations.

However, both the results of model-based and recommendation-based methods are rather fixed, users cannot flexibly adjust their routes in real-time. Meanwhile, the methods

provide poor supports for the collaboration and consultation scheduling process in groups, while a collaborative scheduling method provide users more intuitive and real-time scheduling supports. Therefore, a hybrid method is recommended; users could schedule their routes together on the basis of auto-generated or recommended routes for more practical uses.

B. SELF-DRIVING WITH EDGE COMPUTING

The design of a driverless car could be really complicated; it consists of various modules; like computer vision identification, sensor detection, obstacle avoidance, navigation, and vehicle control modules [12], [19]. All the self-driving related technologies have been studied and developed in their own field for many years, and they have been more and more mature and applicable in real uses, which provides the valuable chance for the rapid development of self-driving technology. And different levels of self-driving technology have already been in use in nowadays driving practices [18].

A self-driving car could probably generate one-gigabyte data every second, and these data should be processed in real time to provide correct commands for the vehicle to drive safely [13]. However, if all these data are uploaded to the cloud and finish their computations at the cloud side, it would bring tremendous pressure to the network bandwidth and reliability, meanwhile results in long latency, which violates the requirements of real-time responses in self-driving scenarios [14], [29].

Therefore, edge computing paradigm is widely adopted in the self-driving scenario [29]. Real-time and urgent services, like navigation and obstacle avoidance services, are processed and provided in real time at the edge side, and these services need to be optimized considering the limited processing and storage capability at the edge side. Taking navigation services as an example, as the car drives in a limited area within a certain period, that only a few map blocks would be stored at the edge side to cut down storage size and processing costs, and such strategy is also adopted in our partial replication scheme.

C. REAL-TIME COLLABORATION AND CONSISTENCY MAINTENANCE

As our work is to implement the real-time routes sharing and scheduling activities under the self-driving scenario, the strong real-time consistency and immediate response time are highly required; we would discuss classical consistency maintenance algorithms and partial replication scheme in this part.

Collaborative editing technique was firstly raised by Ellis and Gibbs [20] in the 1980s, and it's applied to the real-time collaborative editing on documents. Later, a group of researchers devoted to this field, studying about different algorithms [21]–[23]; they adopted different ideas to maintain the consistency; like making transformations on the operations accordingly [20], [23]; recording and retracing the document to its generated state for direct operation executions in the AST scheme [21]; and finding the right execution position by recording its logical previous and next node [22].

Later, researchers started to make optimization of the algorithms on various aspects. The widely adopted vector logical clock has the disadvantage of growing size with the entry of new sites, which could be inefficient in large collaborating environments. Ning *et al.* tried to reduce its size by using the transitivity of causal relation [31], and achieved significant improvements; while in Shao et al.'s work, they focused on cutting down the complexity of transformation of a sequence of accumulated operations [32]. Similarly, in our work, we also tried to make optimization on the accumulated operations' centralized timestamp to reduce transmission costs.

With the maturity of the collaborative editing algorithms, researchers started to extend collaborative editing techniques to more data models, like tables [24], 3D models [25], routes planning model [2], and also some promotions under special scenarios like supports on string wise operations [26], partial replicated tree structures for mobile comments scenario [27].

As our work is to support real-time routes sharing and synchronizing activity. Traditional lock-based methods might greatly influence the response time and affect user experience. The design of address space transformation [21] algorithm provides a lock free collaborative editing pattern, that users could make operations on their local sites without locks, and it supports making synchronizations with other collaborating sites correctly regardless of the receive order of remote operations, which helps to hide the network delay. Besides, they continued to optimize their algorithm by partial replication [27] and reducing the size of timestamps [31], which perfectly resolves the challenges under our highly dynamic scenario, that all vehicles might generate a large number of operations in the real time, causing a high response time and large data transmission. Therefore, we developed our model on the basis of an AST-based collaborative itinerary planning model [2]. We profited from the partial replication idea in mobile commenting [27] and limited local map blocks in edge computing [14], that the car just needs to synchronize with other routes that are currently within the same area, which could greatly reduce the unnecessary synchronizing data. Besides, we proposed a snapshot scheme and compression on consecutive operations to further cut down the processing and transmission costs.

III. PROBLEM AND MODEL DEFINITION

A. ITINERARY PLANNING UNDER SELF-DRIVING SCENARIO

Real-time itinerary sharing and scheduling in self-driving is a relatively new demand, and we would make detailed illustrations and clear definitions of it in this section.

Self-driving tours are becoming increasingly popular these years; a group of people rent or drive their own cars to travel to remote areas for tours. And in such remote areas, routes planning and real time information sharing seems to be extremely important, as the navigation system might not cover all the

unknown spots and paths in the remote areas. Meanwhile, there could be unexpected traffic accidents or terrible road conditions; assisted scheduling from other experienced or leading drivers could be really helpful. Benefiting from the rapid development of self-driving technology, scheduling while driving would become technically practicable in the near future. Besides, anonymous routes sharing with sound safety and privacy-protect mechanisms could provide the cloud side more information to make better traffic controls, which is also worth expecting for future traffic conditions.

We summarized and extracted the required features of a technical scheme that could possibly fit in the needs of the all above scenarios.

1. Intuitive; the routes should be presented on a map, that users could make operations on the routes and notice the changes directly; meanwhile, all the changes could be updated and synchronized in real time for better collaboration experience.

2. Highly responsive; as the vehicles are driving at a high speed, that the current position changes frequently on the map, which generates a large number of operations in a short time. Similarly, the other vehicles also generate a huge number of operations; all these operations should be synchronized and processed at the edge side in an efficient way.

3. Low transmission cost; as the vehicles generate a large amount of synchronization information, which could bring huge transmission costs, and the costs might challenge the current mobile network. An optimized scheme should be designed to cut down the transmission costs.

B. TECHNICAL CHALLENGES AND SOLUTIONS

To build a system that could fit in the above-mentioned features, we have to address the following challenges:

1) how to support collaborative editing behaviors under partial replication architecture. The consistency of real-time itinerary planning has already been guaranteed in Yang's work [2], but how to guarantee the consistency under the partial replication scheme? Is it simply a replicated version that differs nothing from Yang's model? It's obviously not. As, the scheme has to maintain the causality of all operations from different active areas, dealing with the specialty of cross-block edge operations, and so on.

2) how to deal with the peak transmission caused by the change of active areas? The partial replication could reduce the real-time transmission cost, but it also accumulates all the history operations. Once moving into a new area, all the history operations still have to be synchronized, which could cause a peak transmission in the network, and possible higher data loss. This is the key problem that we have to address by adopting partial replication.

3) how to incrementally make synchronization of the data?When the users are scheduling on the map, they might slide on the maps to see surrounding conditions, and during the process, the interface might enter the same area for multiple times, if we synchronize all the data, it would cause huge transmission costs to the network and produce unnecessary transmission. Therefore, it's important to design an incremental synchronization method to save bandwidth.

4) how to design a stable system that the vehicles could still function normally when there are sudden network disruptions? Though the network infrastructure has been built quite well in recent years, there are still terrible network conditions in remote areas, and possible network disruptions would happen occasionally. It is of vital importance to guarantee, that the vehicles could always function well regardless of any network conditions.

In the Themis system, we have designed a sound technical scheme to copy with the above challenges. We developed our scheme on a basic AST implemented system [2] to support collaborative scheduling activities, and further improved it with partial replication design. The correctness of the partial replication scheme has been proved and analyzed in section 6. Besides, we cut down the peak transmission by further adopting snapshot strategy to history operations. And the incremental synchronizations could be implemented by storing all the data at the cloud side, while making synchronizations when needed. In facing network disruptions, our system could still function well, that users could operate on their local sites in real-time, while remote operations could always be synchronized correctly regardless of their arrival time. Meanwhile, as the vehicles are controlled by real-time routes scheduling and local obstacle avoidance modules, which don't depend on network connections; the vehicles could run normally regardless of network conditions. However, there would also be some small inconveniences, as the long-time disconnection might introduce conflicts on the same routes, which currently could only be resolved by human operations, and we would develop corresponding strategies in the future work.

C. BASIC AST MODEL

AST algorithm [21] is originally proposed to support the collaborative editing behaviors on a same document. It organizes the documents in the form of character nodes. Each first add operation would create a node, and each node has a visible flag to identify the character's visible state. Operations on the same character would be added to the node's operation list.

In the AST model, each operation is assigned a timestamp, and the comparing rule of causality relations is designed according to its structure. With adopting the same rule, the causality relation of operations and order among concurrent operations could be kept the same on all sites. With the timestamp, the AST algorithm can retrace the document's state to the remote operation's execution time, and find the right and consistent execution position in the node list on all sites by comparing the value of timestamps. Then, the operation could be executed directly on the position. And by modifying the value of timestamp, it could retrace the document to its newest state.

The advantage of AST algorithm is that it is a lock-free consistency maintenance algorithm; the local operations

could be executed directly, and remote operations could be executed correctly when it is received. Users would not sense any pose when making synchronizations. Besides, the correctness of the algorithm has been strictly proved in mathematical methods. The later proposed unique identifier design further improved the algorithm by leaving out the retracing process. The features of lock-free, highly responsive, and senseless on remote synchronization process perfectly match the features of our scenarios. Thus, we chose the AST algorithm as the basis of our model to achieve real-time local responsiveness and correct remote consistency maintenance and made further improvements on it.

D. MODEL DEFINITION AND NOTATIONS

1) BASIC UNITS AND PRIMITIVE OPERATIONS

As in Yang et al.'s scheme [2], they have already built a basic model for itinerary planning on the AST algorithm, so we further developed our scheme on its basis. Points and edges are the basic operating units; the routes could be represented by points and edges. Every point keeps its operation list and its locations, while every edge keeps its operation list and the locations of its two side points. The routes could consist of lots of small edges and points.

As for the operation, we provide three types of basic operations; all the actions like real-time scheduling, location sharing, and area movements could all be made up of the following operations.

1. Add / Delete point; add or delete a point with its location; e.g, add_node(12.32,91.33); (12.32,91.33) is the location information of a point.

2. Add / Delete edge; add or delete an edge with its two side points; e.g, delete_edge((12.32,91.33),(11.98,90.15)); (12.32,91.33) and (11.98,90.15) are the locations of the two side points of the edge.

3. Activate / Dis-activate block; activate or dis-activate a map block with its location to start or end the synchronization; e.g, activate_block(12.32,90.15); (12.32,90.15) is the center location information of a map block.

The scheduling and car movement actions could be achieved by adding and deleting points and edges, while the movement from one map to another, could be realized by activating and dis-activating certain blocks to turn on/off the synchronization within a certain block.

2) BASIC DATA STRUCTURE

All the actions are decomposed into operations, and our synchronization and consistency maintenance are based on operations.

In the basic itinerary planning model [2], at the user side, it keeps an LHB(local history buffer) to temporarily store the local generated operations, and an RHB(remote history buffer) to temporarily store remote operations, and execute the remote operations when it's causally ready. While at the server side, it keeps a VHB(virtual history buffer) to store all the received operations, and send them to different sites for consistency maintenance. Besides, the server side also maintains an SRN (Server Receive Number) value to identify the operation received order.

As for the operation processing, they create a list and visible flag for each point and edge node, adding operations to the corresponding node's list, and adjusting the flag status according to the operation effects, as shown in Figure 2.

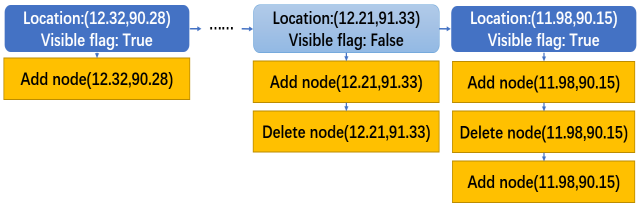

FIGURE 2. Storage form of nodes.

3) TIMESTAMP AND CAUSALITY

Timestamp is the most classical element used to determine the causal relation among operations, and to further guarantee the final consistency. We define an operation A is casually before B, when at the generation of operation B, operation A has already been executed at the B' site, and A's execution effects have already been shown at B's generation time. In this paper, we adopted an optimized centralized timestamp to save transmission costs, and it is defined as;

< siteId, opcnt, lastUpdateSRN, SRN>

siteId is assigned by the center server at the beginning, and it could uniquely identify a site; while opcnt represents the current local site generated operation number. SRN (Server Receive Number) is a unique number assigned by the server to identify the operation's received order at the cloud side, and the SRN value of an operation is kept empty until it is assigned a value from the cloud side. And lastly, the lastUpdateSRN value records the last updated SRN value at the current site when the operation was generated, and it is the key value to determine the causality relation among remote operations, which we would introduce later in section 5.

With the defined timestamp, we could define the causally before relation $\rightarrow$ as follows;

$TS_A \rightarrow TS_B$,

if 1) $TS_{A.siteId} = TS_{B.siteId}$, and $TS_{A.opcnt} < TS_{B.opcnt}$;

or 2) $TS_{A.siteId} != TS_{B.siteId}$, and $TS_{A.SRN} != NULL$, and $TS_{A.SRN} < TS_{B.lastUpdateSRN}$

And if there's no such relation that $TS_A \rightarrow TS_B$, nor relation that $TS_B \rightarrow TS_A$, we define TS_A as concurrent with TS_B, denoted as $TS_A \parallel TS_B$.

4) T-ORDER FOR CONCURRENT OPERATIONS

As the operations are finally added to the queues of point and edge node, the actual operation effect is determined by the order of the operations, thus keeping the same order of operations among all sites is really important for achieving consistency. With timestamp, the order among non-concurrent operations could be arranged, but it's still

unable to determine the order among concurrent operations; thus we introduced other rules to further determine the operation effects of concurrent operations.

T-Order is a rule [21] that could distinguish the order among different concurrent operations, and keeps the order the same at all sites; thus we chose the global unique SRN value as the T-Order comparing value; since SRN is unique within the whole cooperating network, and it is able to keep the same order on different sites.

IV. TECHNICAL SCHEME

As we have analyzed in section 3.A, the designed schema should be intuitive, highly responsive and supporting low transmission costs. Therefore, we adopted typical edge computing architecture shown in Figure 1; storing the current block data at the edge side to guarantee real-time responsiveness, meanwhile storing the whole data at the cloud side to do some calculations to cut down the necessary synchronization data. Under the edge computing architecture, we proposed the partial replication, snapshot, and compression strategies to build a highly efficient model.

A. PARTIAL REPLICATION

We used the same idea in self-driving navigation services [14] and mobile commenting scenario [27], that we only store currently visible map blocks at the edge side, and synchronize and maintain the consistency of current visible blocks to reduce transmission costs and calculating pressure at the edge side, while storing the whole data at the cloud side for possible synchronizations and calculations.

1) EXTENDED STRUCTURES AND NOTATIONS

To implement the partial replication scheme, we would have to add some new data structures and notations both at the edge and the cloud side. And we have listed them below;

Edge Side

1)BlockId; as we are dividing the map into small blocks, we need to assign each block a unique id that could identify the real same area in all sites, and the id is related to the locations of the current area.

2)BlockLastUpdateSRN; the last Updated SRN value of operations within a certain block; as every site keeps synchronizing with the cloud side about only certain current visible blocks, if keeping a single lastUpdateSRN value for each site, it would be impossible to identify the unsynchronized operations for every single block; so every block has to maintain a BlockLastUpdateSRN value separately.

Cloud Side

1)SiteBlockLastUpdateSRN; the lastUpdateSRN value for each block of every site; this value could help distinguish synchronized and unsynchronized operations of each block of different sites when there are synchronization requests.

2)SiteActiveBlock; the data structure helps to record the current active blocks of each site, and it could help to find out the sites that have collaborating relations by checking its value at the cloud side.

3)BlockLastReceiveSRN; the last added SRN value of the operation in this block at the cloud side, which could be used to firstly check if there's the need to search the VHB for unsynchronized operations.

4)BlockOperationList; this data structure stores the operations from different blocks for faster operation search, which is not compulsory, but it can help greatly to reduce the search cost, as operations are synchronized separately in each block.

2) DATA SYNCHRONIZATION UNDER PARTIAL REPLICATED ARCHITECTURE

In the partially replicated architecture, the cloud side listens to all channels from the edge side. Once it receives an operation or an operation list from a site, it would store the operations in its BlockOperationList and send back the assigned SRN values, presented as in Algorithm 1.

Algorithm 1 Process Operation Data Package
Operation List:OP **Server Receive Number**:SRN
BlockLastReceiveSRN:BLRS **BlockOperationList**:BOL

```
set BlockID = OP[0].blockID
set index = 0
repeat
    set op = OP[index]
    Send back SRN value with op.opcnt
    set op.SRN = SRN
    SRN++
    BOL[BlockID].push(op)
    index++
until index equals to OP.length
set BLRS[BlockID] = SRN − 1
```

As for the operation synchronizations, both client-pull and server-push methods are compatible with our partial replication scheme. In the server-push pattern, once the server receives operations from the clients, it would check out the collaborating sites with the sending site, and after assigning the SRN values and storing the operations as in Algorithm 1, it would send the operations to collaborating sites. And when the server receives activate and dis-activate operations of blocks, it would firstly modify the value of SiteActiveBlock, send the accumulated operations to the newly joined blocks if there are accumulated unsynchronized operations.

While under the client-pull pattern, the server side would not transfer operations to collaborating sites in real-time, it would store the operations in corresponding block operation list shown in Algorithm 1, and make synchronizations when it receives synchronization requests from clients. It would find out the unsynchronized operations from other sites and send back the unsynchronized operations to the requesting site, shown in Algorithm 2.

As for the edge side, it only needs to execute local operations and pack them up; sending them to the cloud side at certain synchronizing frequency, meanwhile synchronizing and executing the limited operations that have already been

Algorithm 2 Process Synchronization Request
BlockID:blockID **SiteBlockLastUpdateSRN**:lastSRN
BlockLastReceiveSRN:BLRS **BlockOperationList**:BOL

```
if lastSRN[blockID] < BLRS[block] then
  set index = BOL.size − 1
  repeat
    if BOL[blockID][index].timestamp.SRN >
    lastSRN[blockID] and
    BOL[blockID][index].timestamp.siteID! = blockID
    then
      ResDataList.push(BOL[index])
      index − −
    end if
  until index equals to −1 or
  BOL[blockID][index].timestamp.SRN <=
  lastSRN[blockID]
  Send back ResDataList
  set lastSRN[blockID] = BLRS[blockID]
else
  NOOP
end if
```

filtered at the cloud side, which could greatly reduce the transmission cost and responsive time.

B. APPLYING SNAPSHOT TO HISTORY OPERATIONS

Though the replicated architecture could greatly reduce the real-time data transmission cost, and improve response time, when the operating area moves into a new block, it still have to synchronize all the operations within that block, which might result in high latency and peak transmission within a short period. Sending the accumulated data in batches might alleviate the peak transmission, but the total transmission size won't change and the it might need quite a while to process the accumulated data to synchronize the user interface to the newest state.

However, as shown in Figure 2. The node state is only determined by the execution effects of the last operation, as the execution effects of previous operations are covered by the execution effects of the operation ranking last in the operation list. (e.g, as for the node(11.98,90.15) in Figure 2, the node's visible state is changed from "True", to "False", and to "True" state again, with the addition of add, delete, and add operations). Thus, we could make improvements on this property.

1) SNAPSHOT DESIGN

As we have analyzed that, the final state of a point or edge node is determined by the operation that currently ranged last in the node's list. But simply taking the last operation to replace all the history operations could probably cause problems. As shown in Figure 3, if we simply take the last unsynchronized Operation of Node("12.32,90.28") to replace all the previous operations, the last delete operation

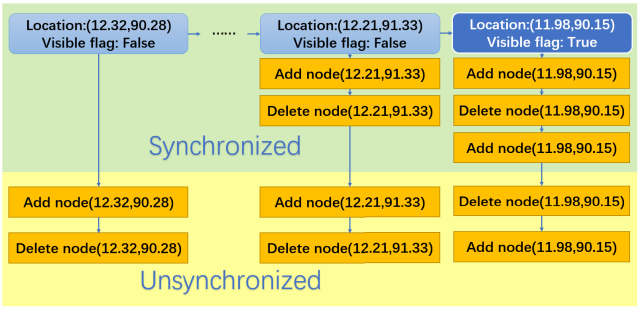

FIGURE 3. A sample of the application of snapshot strategy.

would try to delete a non-existing node; while for the Node("12.21,91.33"), the last delete operation would try to delete an invisible node, and for the Node(11.98,90.15), the last add operation would try to add an already existing node.

However, these possible exceptions won't change the operation's last effective property; the problem could be well addressed by introducing a complete error-torrent mechanism at the edge side. We have listed all the possible conditions with the operation type and node's state in Table 1. As for the abnormal conditions (also shown in Figure 3) that could cause errors, we have designed error handling-solutions shown in Table 2.

TABLE 1. All the possible conditions by adopting snapshot strategy.

Operation Type \ Node's State	Non-existing	Visible	Invisible
Add	Normal	Error	Normal
Delete	Error	Normal	Error

TABLE 2. Error conditions and coping strategies.

No	Condition	Strategy
1	Deleting a non-existing node.	Create a corresponding node with the timestamp of the delete operation, add the delete operation, and set the visible flag to false.
2	Deleting an invisible node.	Add the delete operation to the node operation list.
3	Adding an already-existing node.	Add the add operation to the node operation list.

As for the first condition of deleting a non-existing node, firstly we have to guarantee that the node exists, and the node's state is currently invisible, and more importantly, the invisible state of the node is determined by the current delete operation, that all the effects of the causal before and concurrent operations with smaller T-Order are ineffective.

Therefore, the strategy is to create a node, add the delete operation with its timestamp to the operation list, and set the node to the invisible state. As for the second condition, the creation of nodes could be omitted, while adding the operation with its timestamp to the operation list to maintain the causal and T-Order among operations on all sites, and the same strategy could also be applied to the third condition. More detailed analysis and proof could be found in section 5.B.

With the proposed error coping strategies, the three types of error shown in Figure 3 could be well resolved. The deletion of node (12.32,90.23) would no longer try to delete a non-existing node, but try to add a node and set the node's state to invisible instead, and the deletion of node (12.21,91.33) would not delete an invisible state, but simply adding the delete operation to its operation list, and the addition of node (11.98,90.15) would also avoid to add an already existed node, but simply put the add operation to the list.

C. OPTIMIZATION ON CONSECUTIVE OPERATIONS

Furthermore, when there's only a single active site (user) within its current zone, the site doesn't have to synchronize with the cloud side so frequently (the map could be divided into small blocks), as there are no other sites could affect its current routes, and its routes won't affect other active sites in other blocks either. Besides, as human beings could only sense limited changes within a certain period of time, raising the synchronizing frequency to a certain level won't make obvious differences to a single user's perception. Thus, we could make optimization on these accumulated operations to reduce transmission costs.

1) COMPRESSION OF CONSECUTIVE OPERATION

As we have analyzed that, a site doesn't have to synchronize with the cloud in real-time; the operations could be accumulated and share a consecutive timestamp to reduce the transmission cost. As the opcnt value increases continuously for consecutive operations, and lastUpdateSRN value keeps the same if there are no synchronization actions within the time of the operation sequence. Therefore, as for a consecutive and non-synchronization interrupted operation sequence, they could share a timestamp together. And later, at the cloud side, the timestamp could be parsed and reassigned to each operation. So, the edge side could keep synchronizing with the cloud side at a certain frequency, and within the period, the uninterrupted consecutive operations could share the timestamp to make data compression.

2) SIMPLIFICATION OF CAR MOVEMENTS

Besides, as for the operations produced by car movements, they could be stored in a temporary queue, when there's the need for synchronization, simply taking the first and the last operation in both point and edge queue could represent the movement track of the car within the time. However, such optimization might cause information loss, as the slight changes in directions might be lost. Therefore, the number of operations to be simplified each time should be controlled at a certain range, that the simplified route doesn't vary too much from the previous one, and it won't violate real physical road conditions.

V. ANALYSIS OF CORRECTNESS

A. CAUSALITY MAINTENANCE UNDER PARTIAL REPLICATION

Timestamp is used to identify the causal relation among operations. As for the centralized timestamp < siteId, opcnt, lastUpdateSRN, SRN > design, the casual relations among operations from the same site could be easily identified by the opcnt value, as the opcnt value increases with the generation of operations; thus, later generated operations are bound to have larger opcnt values than their previous operations. As for operations from different sites, if $OP_{A.lastUpdateSRN} > OP_{B.SRN}$, which means at the generation of operation A at its local site, operation B's effect has already been shown on A' site, so we can identify that operation B is causally before operation A; as for operations having no casually before relation, we define these operations as concurrent operations.

In the partial replication scheme, if we still keep a single lastUpdateSRN value for the whole site, when the active blocks change, we would be unable to locate unsynchronized operations for the new block as the lastUpdateSRN value might have skipped many operations, which could cause the loss of the synchronization of many operations. Considering that each block actually only synchronizes with the operations generated at those blocks with same locations of other sites; blocks with different locations are actually keeping isolated with each other on the causal relations of operations. Therefore, we just need to maintain the causal relations within the blocks sharing the same blockID. Extending the lastUpdateSRN value to each map block, and maintaining the lastUpdateSRN value for each block could address the problem. We could regard each block as an independent site, and the site with same blockIDs would make synchronizations within a group. Though all the blocks share the same SRN value, the SRN value might be nonconsecutive within a block's operation, but it can still represent the receiver order of the operations, and the causal relations could be correctly maintained within each block.

B. ACHIEVING SAME EFFECTS BY ADOPTING SNAPSHOT STRATEGY

The correctness of applying the snapshot technique to the history operations could be easily proved. To prove the correctness of taking the last operation as an alternative of all the history operations, we just need to ensure the correctness of the four following issues;

1. Is the effect of last operation same with all the effects of history operations? As our operations are added to each node, and each node's visible state is determined by the visible flag, which could only be changed by add and delete operations. Meanwhile, both the add and delete operation change the visible flag in the form of coverage, which guarantees that

the node's visible state is only determined by the operation that ranks last in order.

2. How does the current scheme guarantee selecting the same operation as the last one on all sites? As we have defined the order of the non-concurrent operations by comparing timestamps and concurrent operations by T-Order, all operations in all sites would observe the same order, so our scheme could guarantee to select the same ranked last operation on all sites.

3. What if operations that rank before the last operation, are synchronized after the current last operation? As for the operations within a synchronization period, we only send the last operation in the unsynchronized list, so there would be no previous operations reach behind the current last operation in the current unsynchronized operation list. But there could be possible synchronizing operations blocked in the previous synchronization period, and for such operations, we could compare its order with the latest operation, and if its order is before the current latest operation, it could be added to the node's list without doing any actions.

4. What if the operation list is synchronized in several times, and the effect of the current last operation contradicted the previous unsynchronized node state? We have listed all the possible contradicting conditions in Table 2, and designed corresponding error tolerance strategies. As for deleting a non-existed node, which our scheme choose to create a node with the timestamp of the delete operation, and set the node visible state to false. In this way, it reaches the same execution effects of adding and deleting operation pairs. Any previous operations are no longer effective, which we have proved in issue 3. And for adding an already existing node, adding the operation to node's list could update the node's newest timestamp, and achieve the same effects, and similar strategies for deleting an already deleted node.

C. CORRECTNESS OF CONSECUTIVE OPERATION COMPRESSION

As for the accumulated operations within a certain period, these operations share the same lastUpdateSRN value and consecutive opcnt value, and their timestamp could be compressed into one, reducing the transmission costs.

The key point in accumulating operations is that different sites don't have to synchronize with each other in real time, that they could tolerate a certain synchronization ratio, like 100ms every time. As long as we keep the synchronization period at a certain level that, the delay of synchronizations are hard to be sensed by human beings, the accumulations of operations won't affect user experience.

And the only thing we have to prove is that the accumulation of operations doesn't affect the final consistency on different sites. The proof process is quite similar to the snapshot part, as the final state is determined by the last operation in the list which we have already analyzed. The strategy of accumulating operations might turn casual-before relation into concurrent relation among different sites, as the synchronization has been actually delayed, operations that might be causal before relation, now could be concurrent by adopting consecutive compression strategy. However, the order among operations could still be kept the same at all sites by adopting the timestamp and T-order design, that the final consistency could still be reached.

As the changes from causal relation to concurrent relation doesn't really matter. The possible consequences of delay might cause deleting the same point or edge, deleting an already not existed node, and adding the same point or edge, which actually could also happen in real-time synchronizations. As for the above conditions, we could adopt the coping strategies in Table 2 and guarantee consistency.

As the simplification of taking the first and the last operation in both point and edge list to represent the movement of the vehicle within the time, which is easy to understand. As the two operations achieve the same effect as a sequence of operations, and all these operations are within an unsynchronized period, simplifying the sequence of operations into two operations won't affect final consistency.

D. DEALING WITH CROSS-BLOCK EDGE OPERATION

In fact, in the proof of the adjusted timestamp, we made the hypothesis that all the points and edges are within each block, that the operations from different sites are totally isolated. However, there would certainly be cross-block edges when the routes cut cross blocks. If we still adopt the current scheme, active users on different blocks might operate on the same edge, which might break the isolated causality of the blocks, and bring possible errors.

As for the condition, we could assign a new virtual block for every two adjacent blocks, assigning these edges to the virtual blocks, and adopting the same rules for these blocks; treating them the same as other blocks; and keeping them synchronized independently within each block, and the causality of these cross-block edge operations could be correctly maintained within these virtual blocks.

VI. EXPERIMENTS AND PERFORMANCE

To evaluate our scheme, we implemented a system called Themis and the baseline system CLIP in Yang's study [2]. The two systems shared most of the codes, while we differed them by setting flags for different optimization strategies. With setting the partial flag, we could turn on/off the partial replication strategy and compare the performances, and the same for other optimizing strategies. The web communication is achieved by web socket approach, and we adopted the client-pull pattern, that the edge side fetches synchronization data packages from the cloud side at a certain frequency; sending operation data packages in real-time or accumulating the operations when testing the optimization of consecutive operations. We implemented the routes scheduling function by adopting Baidu Map APIs; as shown in Figure 4, users could operate on the routes and maps directly with adding, deleting and sliding operations. The experiment was conducted on a CPU i7-7700 3.5GHz, main memory 16G, windows 10 system machine.

FIGURE 4. User interactive interface.

We compared the real-time data transmission and responsive time of remote operations under partial replicated and non-partial replicated conditions, the execution time of every 1000 remote operations with and without snapshot strategy, and the compression ratio of consecutive timestamps.

A. PARTIAL AND NON-PARTIAL REPLICATION SCHEME

We profited the partial replication strategy mainly from the partial mobile commenting scenario, that a site only needs to synchronize with the server side about its current operating areas, and synchronize with other areas if the operating area changes.

To conduct the experiments, we firstly define the partial replication ratio as,

$$\frac{\sum_{i=1}^{n} Ncs_i}{N_{total} * n}$$

The NCS_i means that the number of sites that is currently located within the same area with i^{th} site, and N_{total} represents the total sites number that the i^{th} site has to synchronize with if not adopting the partial replicate strategy, and n represents the number of sites that has collaborating sites within its area.

Taking partial replicate ratio of 70% as an example, we set the number of total sites to 11, and we hypothesized that the operating area of each site is limited within one block, and the locations of its operating areas are shown in Figure 5. Among the 11 sites, there are 8 sites that have collaborating sites, so the value of n is 8. As for the sites 1 to 8, they all have 7 collaborating sites, so the value of the partial replication ratio is 70%.

To compare the effects of the partial replication strategy, we locate the sites in the same areas when testing the partial replicated and non-partial replicated scheme; setting an

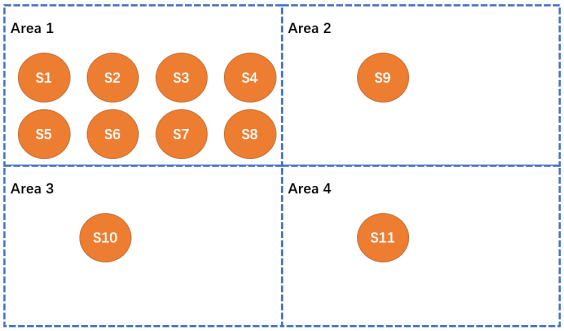

FIGURE 5. Locating positions of simulating a 70% partial replication ratio.

synchronizing period at 50 milliseconds. And we evaluated the optimization effects by only comparing the real-time transmission data size and the average execution time of remote operations of sites that have collaborating sites within its current block; as for the sites in Figure 5, the evaluated sites are numbered from 1 to 8.

We kept the total site number at 11 for similar testing pressure of all conditions, and set the partial replication ratio from 10% to 100%. And we simulated human operations by generating one node or edge operations every second on each site, and the node and edge operations each take up 50%(as the routes are usually presented in a single path, and the number of nodes is roughly the same as the number of edges in a route).

We ran each test 5 times to avoid possible errors, and got the real-time data transmission size and the average remote operation execution time of partial replicated and non-partial replicated scheme with different partial replication ratio shown in Figures 6 and 7. As we could see, that the real-time data transmission size and the execution

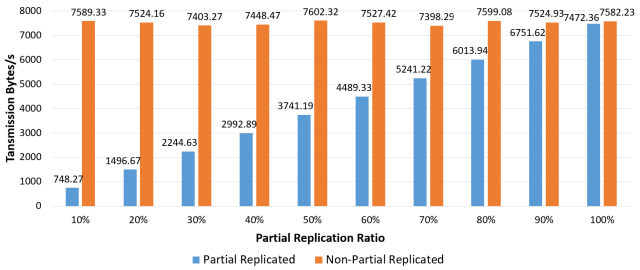

FIGURE 6. Size of transmission data with and without partial replication.

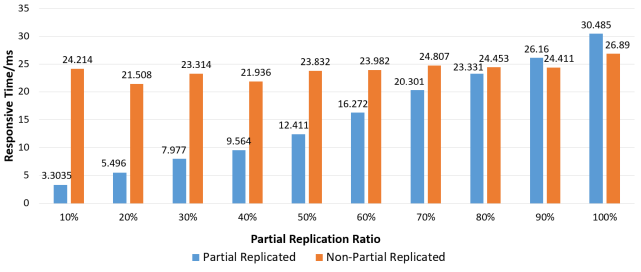

FIGURE 7. Average responsive time of remote operations with and without partial replication.

time of remote operation of non-partial replicated scheme are rather stable, as each site actually keeps synchronization with another 10 sites in all test conditions. The real-time data transmission size of partially replicated scheme could be significantly reduced with the decrease of partial replication ratio, and the data sizes are always less than or equal to the non-partial replicated ones. And the execution time of remote operations in the partially replicated scheme is less than the time of non-partial replicated scheme in most conditions, and a little bit higher when the partial replication ratio reaches 90%. Theoretically speaking, the expected average execution time of partial replication scheme is supposed to be always less than or equal to the non-partial replicated scheme, as there are no too many differences at the execution logic on the edge side. The higher performances at 90% and 100% could be possibly explained by the extra calculating and processing costs at the server side, as we ran the experiments on a single machine.

However, it's clearly shown in Figures 6 and 7 that, with the decrease of the partial replicate ratio, our scheme could significantly reduce the real-time data transmission size and the responsive time of remote operations comparing with the non-partially replicated scheme. And in real conditions, all the cars are actually located very sparsely on maps, and each car just need to keep synchronizing with very limited cars, keeping the partial replication ratio at a really low level, and apparently under such conditions our scheme could greatly reduce both the transmission and processing costs, reducing network congestion, improving the responsive speed of remote operations, and our scheme perfectly matches the idea of reducing the unnecessary calculations and transmission at the edge side, while leaving more complicated and non-highly responsive required calculations at the cloud side.

B. SNAPSHOTTED AND NON-SNAPSHOTTED COMPARISON

As we have proved in section 5.B, taking the last operation on the same node is able to represent all the execution effects of all its previous operations. As for testing the optimization effects of the snapshot strategy, we define the Repeated Ratio as,

$$\frac{N_{added}}{N_{total}}$$

N_{added} represents the operations that are added up to already existed edge or point node, while N_{total} represents the total number of operations.

As for the experiment, we set two sites located at different blocks, one of them generated 1000 operations with increasing repeated ratio, and we recorded the execution time of all these 1000 remote operations when the other site moved into the same area.

As shown in Figure 8, with the growth of the repeated ratio, both the snapshotted and non-snapshotted scheme has cut down its execution time, which is because, with the growth of operations, there would be more operations executed on

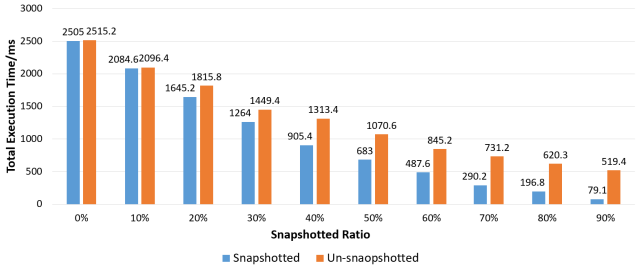

FIGURE 8. Execution time of 1000 operations.

already existed nodes; there would be less creation of point and edge nodes and other related data structures, and these operations could be executed directly by changing the visibility and adding the operations to the operation list, which certainly cuts down the cost even without adopting the snapshot strategy.

However, we could see that the snapshotted scheme costs less than the non-snapshotted scheme in all conditions. And to further indicate the compression ratio changes of the snapshotted scheme of the non-snapshotted one, we presented the proportion of the snapshotted one occupying of the non-snapshotted one in Figure 9. We could see that though both the snapshotted scheme and non-snapshotted scheme are cutting down its execution time with the increment of repeated ratio, the snapshotted scheme could continuously achieve a higher compression ratio on the non-snapshotted scheme with the repeated ratio goes up.

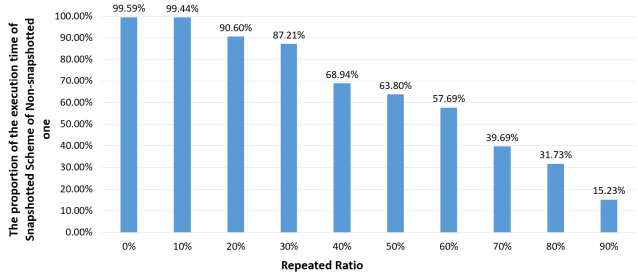

FIGURE 9. Compression ratio of the time cost.

As for the application of the snapshot strategy, we could notice in Figure 8 that, the snapshotted scheme performs all the better or equal to the non-snapshotted one regardless of the repeated ratio. Meanwhile, the filtering of the last effective operation in a list might take some costs, but the process is done at the cloud side, adding no extra costs to the edge side. Besides, the snapshotted scheme could apparently cut down the transmission costs, as the transmission data becomes smaller, which we would make no more extra experiments to prove in this part.

C. IMPROVEMENT ON CONSECUTIVE OPERATIONS

The experiment of compression on consecutive operations is rather simple. We sent different number of operations using

unique timestamp and shared timestamp, and we captured the data package to analyze its size.

As shown in Figure 10, the shared timestamp scheme could always achieve a compression ratio of more than 22.5% when there are more than 10 accumulated operations, and the compression ratio reaches its upper limit of about 25.0% when there are more than 100 operations. The compression upper limit is actually determined by the proportion of the size of a timestamp taking up within in a complete operation, and the compression ratio would reach a limit with the compressed operation number goes on. So, to make a trade-off of transmission cost between synchronization frequency, we could probably choose to make a synchronization when there are 100 operations or when it reaches the synchronization period.

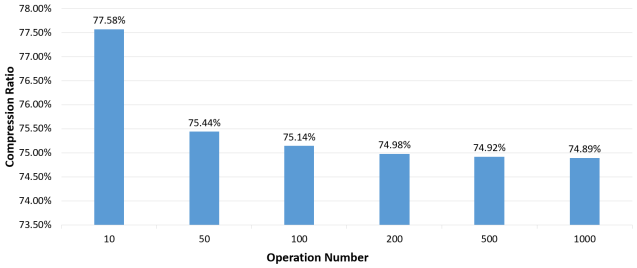

FIGURE 10. Compression ratio of consecutive operations.

As for the strategy of taking the first and last operation to represent the movement of a vehicle is rather simple and straight, and its optimization on transmission cost could be easily modeled as 2/n of its original size(n represents the total operation number within the period), we would make no further experimental proof here.

VII. CONCLUSION

In this paper, we introduced and illustrated several new routes sharing and synchronizing needs under the self-driving scenario. We proposed and implemented the Themis system to support the needs, making optimization by adopting partial replication, snapshot, and consecutive compression strategies; and the correctness of all these strategies have been analyzed and proved in our paper; the Themis system has achieved significant improvements of performances on all various aspects compared with Yang's basic scheme [2].

Besides, our scheme made further optimization on the partial replication approach by taking snapshot on covering-effect editing objects. The partial replication design and consecutive compression strategy could be easily generalized and applied to other large scale collaborative editing scenarios, and the last-effective snapshot and error-tolerant design could also be used on other forms of collaborative object editing with covering effects.

Our technical scheme could well support scheduling while driving practices, providing better driving and traveling experiences. Besides, the scheme could also be generalized to more scenarios like routes scheduling and sharing of search and rescue robots; the control of AGV(Auto Guidance Vehicle).

In the future, we would continue to optimize our system by developing coping strategies to resolve conflicts introduced by long-time disconnection, taking snapshots of local operations to reduce local data storage, and making more compression of operations at the cloud side to further reduce the transmission cost. Meanwhile, we would evaluate our system with more comprehensive experiments on various performances, and try to integrate our technical scheme into self-driving technologies.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their insightful, and detailed feedback, which improves the presentation and understanding of this work.

REFERENCES

[1] B. Schoettle and M. Sivak, "A survey of public opinion about autonomous and self-driving vehicles in the U.S., the U.K., and Australia," Transp. Res. Inst., Univ. Michigan, Ann Arbor, MI, USA, Tech. Rep., Jul. 2014.

[2] D. Yang, T. Lu, H. Xia, B. Shao, and G. Ning, "Making itinerary planning collaborative: An AST-based approach," in *Proc. CSCWD*, NanChang, China, May 2016, pp. 257–262.

[3] A. Gunawan, H. C. Lau, P. Vansteenwegen, and K. Lu, "Well-tuned algorithms for the team orienteering problem with time windows," *J. Oper. Res. Soc.*, vol. 68, no. 1, pp. 861–876, Dec. 2017.

[4] H. Hashimoto, M. Yagiura, and T. Ibaraki, "An iterated local search algorithm for the time-dependent vehicle routing problem with time windows," *Discrete Optim.*, vol. 5, no. 2, pp. 434–456, May 2008.

[5] L. Ke, C. Archetti, and Z. Feng, "Ants can solve the team orienteering problem," *Comput. Ind. Eng.*, vol. 54, no. 3, pp. 648–665, Apr. 2008.

[6] C. Verbeeck, K. Sörensen, E. H. Aghezzaf, and P. Vansteenwegen, "A fast solution method for the time-dependent orienteering problem," *Eur. J. Oper. Res.*, vol. 236, no. 2, pp. 419–432, Jul. 2014.

[7] Z. Luo, B. Cheang, A. Lim, and W. Zhu, "An adaptive ejection pool with toggle-rule diversification approach for the capacitated team orienteering problem," *Eur. J. Oper. Res.*, vol. 229, no. 3, pp. 673–682, Sep. 2013.

[8] S. Boussier, D. Feillet, and M. Gendreau, "An exact algorithm for team orienteering problems," *Quart. J. Oper. Res.*, vol. 5, no. 3, pp. 211–230, Sep. 2007.

[9] I.-M. Chao, B. L. Golden, and E. A. Wasil, "A fast and effective heuristic for the orienteering problem," *Eur. J. Oper. Res.*, vol. 88, no. 3, pp. 475–489, Feb. 1996.

[10] C. Ge, J. Luo, and W. Xin, "Personalized travel route recommendation using collaborative filtering based on GPS trajectories," *Int. J. Digit. Earth*, vol. 11, no. 12, pp. 1–24, 2018.

[11] Y. Sun, H. Fan, M. Bakillah, and A. Zipf, "Road-based travel recommendation using geo-tagged images," *Comput. Environ. Urban Syst.*, vol. 53, pp. 110–122, Sep. 2015.

[12] T. Banerjee, S. Bose, A. Chakraborty, T. Samadder, B. Kumar, and T. Rana, "Self driving cars: A peep into the future," in *Proc. 8th Annu. IEMECON*, Bangkok, Thailand, Aug. 2017, pp. 33–38.

[13] (Jul. 19, 2019). *Self-Driving Cars will Create 2 Petabytes of Data, What are the Big Data Opportunities for the Car Industry*, Accessed: Oct. 2019. [Online]. Available: https://datafloq.com/read/self-driving-cars-create-2-petabytes-data-annually/172

[14] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[15] P. Bansal and K. M. Kockelman, "Are we ready to embrace connected and self-driving vehicles? A case study of Texans," *Transportation*, vol. 45, no. 2, pp. 641–675, Mar. 2018.

[16] *Google has Developed a Self-Driving Car*, Accessed: Oct. 2019. [Online]. Available: https://www.iflscience.com/technology/google-has-developed-self-driving-car/

[17] (May 18, 2019). *How Google's Self-Driving Car Will Change Everything*, Accessed: Oct. 2019. [Online]. Available: https://www.investopedia.com/articles/investing/052014/how-googles-selfdriving-car-will-change-everything.asp

[18] A. Meyrav. (Jun. 19, 2019). *Self-Driving Cars: The Biggest Development in Transportation Since the Automobile*, Accessed: Oct. 2019. [Online]. Available: https://www.etoro.com/blog/market-insights/self-driving-cars-the-biggest-development-in-transportation-since-the-automobile/
[19] S. Behere and M. Torngren, "A functional architecture for autonomous driving," in *Proc. 1st Int. Workshop Automot. Softw. Archit. (WASA)*, Montreal, QC, Canada, May 2015, pp. 3–10.
[20] C. A. Ellis and S. J. Gibbs, "Concurrency control in groupware systems," *ACM SIGMOD Rec.*, vol. 18, no. 2, pp. 399–407, Jun. 1989.
[21] N. Gu, J. Yang, and Q. Zhang, "Consistency maintenance based on the mark & retrace technique in groupware systems," in *Proc. Int. ACM SIGGROUP Conf. Supporting Group Work*, Sanibel Island, FL, USA, Nov. 2005, pp. 264–273.
[22] G. Oster, P. Urso, P. Molli, and A. Imine, "Data consistency for P2P collaborative editing," in *Proc. 20th Anniversary Conf. Comput. Supported Cooperat. Work (CSCW)*, Banff, AB, Canada, Nov. 2006, pp. 259–268.
[23] C. A. Ellis and C. Sun, "Operational transformation in real-time group editors: Issues, algorithms, and achievements," in *Proc. CSCW*, Seattle, WA, USA, Nov. 1998, pp. 59–68.
[24] C. Sun, H. Wen, and H. Fan, "Operational transformation for orthogonal conflict resolution in real-time collaborative 2D editing systems," in *Proc. CSCW*, Seattle, WA, USA, Feb. 2012, pp. 1391–1400.
[25] Agustina and C. Sun, "Dependency-conflict detection in real-time collaborative 3D design systems," in *Proc. CSCW*, San Antonio, TX, USA, Feb. 2013, pp. 715–728.
[26] J. Zhang, T. Lu, H. Xia, B. Shao, and N. Gu, "ASTS: A string-wise address space transformation algorithm for real-time collaborative editing," in *Proc. IEEE 21st Int. Conf. CSCWD*, Wellington, New Zealand, Apr. 2017, pp. 162–167.
[27] H. Xia, T. Lu, B. Shao, G. Li, X. Ding, and N. Gu, "A partial replication approach for anywhere anytime mobile commenting," in *Proc. 17th ACM Conf. CSCW Social Comput.*, Baltimore, MD, USA, Feb. 2014, pp. 530–541.
[28] L. Chang, W. Sun, and W. Zhang, "Review of tourism route planning," *CAAI Trans. Intell. Syst.*, vol. 14, no. 1, pp. 82–92, 2019.
[29] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," ETSI, Sophia Antipolis, France, White Paper 11, pp. 1–16, 2015, vol. 11, no. 11.
[30] W. Souffriau, P. Vansteenwegen, J. Vertommen, G. V. Berghe, and D. Van Oudheusden, "A personalized tourist trip design algorithm for mobile tourist guides," *Appl. Artif. Intell.*, vol. 22, no. 10, pp. 964–985, 2008.
[31] G. Ning, Q. Zhang, J. Yang, and Y. Wei, "DCV: A causality detection approach for large-scale dynamic collaboration environments," in *Proc. Int. ACM Conf. Supporting Group Work*, Sanibel Island, FL, USA, Nov. 2007, pp. 157–166.
[32] B. Shao, L. Du, and G. Ning, "A sequence transformation algorithm for supporting cooperative work on mobile devices," in *Proc. CSCW*, Savannah, GA, USA, Feb. 2010, pp. 159–168.
[33] A. Silberschatz and Z. Kedem, "Consistency in hierarchical database systems," *J. ACM*, vol. 27, no. 1, pp. 72–80, Jan. 1980.
[34] T. A. Funkhouser, "RING: A client-server system for multi-user virtual environments," in *Proc. SID*, Monterey, CA, USA, Apr. 1995, p. 85.
[35] O. Ulusoy and G. G. Belford, "Real-time lock-based concurrency control in distributed database systems," in *Proc. ICDCS*, Yokohama, Japan, Jun. 1992, pp. 136–143.
[36] S. Zhao, D. Li, T. Lu, and N. Gu, "Back to the future: A hybrid approach to transparent sharing of video games over the Internet in real time," in *Proc. CSCW*, Hangzhou, China, Mar. 2011, pp. 187–196.
[37] H. Chen, L. Chen, and G.-C. Chen, "Effects of local-lag mechanism on task performance in a desktop CVE system," *J. Comput. Sci. Technol.*, vol. 20, no. 3, pp. 396–401, May 2005.

TE JIANG received the bachelor's degree in Internet of Things engineering from the Zhejiang University of Technology, in 2017. He is currently pursuing the master's degree in computer science with Fudan University, Shanghai, China.

His current research interests include collaborative computing and social computing.

TUN LU received the Ph.D. degree in computer science from Sichuan University, in 2006. He was a Visiting Scholar with the HCI Institute, Carnegie Mellon University, in 2015. He is currently an Associate Professor with the School of Computer Science, Fudan University.

He has published more than 60 peer-reviewed publications in prestigious journals and conferences such as CSCW, CHI, WWW, NIPS, UbiComp, and so on. His research interests include computer supported cooperative works (CSCW), social computing, and human–computer interaction (HCI). He shared a best paper award at CSCW'15 and an Honorable Mention Award at CSCW'18. He is a Senior Member of China Computer Federation (CCF) and a member of ACM. He is a Standing Committee Member of CCF Technical Committee of Cooperative Computing. He has been active in professional services by serving as a PC members (e.g., GROUP'18, CRIWG'17 & 2018, CSCWD'16), a PC Co-Chairs (e.g. ChineseCSCW'17 & 18, CSCWD'10), an Associate Chairs (e.g. CHI'19 & 20, CSCW'19 & 20), a Guest Editors (e.g. *International Journal of Cooperative Information Systems*), and a Reviewers for many well-known journals and conferences.

NING GU received the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, China, 1995. He is currently a Professor and the Director of the Cooperative Information and Systems Laboratory School of Computer Science, Fudan University, China.

His current research interests include CSCW and collaborative computing, social computing, and human computer interaction.