マルチメディア通信と分散処理　64−23
（１９９４．　３．　４）

移動体間の資源共有方式

濱田 賢、滝沢 誠
東京電機大学理工学部経営工学科

現在の鉄道，飛行機等の交通システムでは，列車，飛行機等の移動体の運行計画が集中的に静的に決定されている．運行計画の決定は，考慮すべき要因が多く，複雑で，大きな計算量を必要とする．このために，自然災害，事故等により，計画通りに運行できない場合に，計画の修正，変更は容易ではない．本論文では，運行計画を分散的に，かつ動的に決定する方式について考察する．移動体のオブジェクトの移動は，原子的，即ち移動できるかできないかと考える．これを，移動空間内のオブジェクトをロックするトランザクションとしてモデル化する．移動体のトランザクションがいつロックを解放するかにより，開，半開，閉の三つの方式を示す．また，移動体の移動形式とロックの関連について論じる．最後に，ロックされるオブジェクト数について，三方式を評価する．

Management of Vehicle Transactions

Satoshi Hamada and Makoto Takizawa
Tokyo Denki University
Ishizaka, Hatoyama, Saitama 350−03, Japan
e−mail {hama, taki}@takilab.k.dendai.ac.jp

This paper discusses how to find a path in a space for a vehicle to move to the destination. The space is composed of space objects structured in a hierarchical tree named a *space tree*. Higher-level objects denote broader area than lower-level ones in the space tree. Vehicles first finds *broader* paths which include higher-level objects in the space tree, and then the path is made more detailed as the vehicles are approaching to the destinations. A movement of each vehicle v on an object o is *atomic*, i.e. v can either pass o or not. Vehicles are modeled as transactions which hold objects in the space tree. There are kinds of vehicles with respect to how to move on the space. For example, cars can stop but airplane cannot, packets can be removed but trains cannot, and cars can back but cannot in the motorway. In this paper, we discuss three schemes, i.e. *close*, *semi-open*, and *open* ones on when the vehicles release the objects, and discuss how the schemes can be adopted to the vehicle types and the evaluation of the schemes.

1 Introduction

A *vehicle system* is a model of systems where something is moving, e.g. packets in communication networks, automobile cars in roads, and trains in rail ways. The vehicle system is composed of *vehicles* which move around in the *vehicle space*. [2, 5, 13] discuss a *broad path decision* where each vehicle v first decides the *broad* path to the destination, and then the path is more detailed as v is approaching to the destination. The space is composed of paths, e.g. roads and rail ways. The space is partitioned into disjoint units named *space objects*. Each object includes some parts of the paths. The objects are further partitioned into smaller objects. Thus, the space is structured in a tree whose nodes denote objects. Each object o monitors the states of the component objects, e.g. how much they are congested.

Since each object o can admit a limited number of vehicles, some vehicle cannot pass o if o is congested. There are two approaches to resolving conflict among the requests on o from multiple vehicles, i.e. scheduling and dynamic resolution. In the scheduling method, the movements of vehicles on objects are scheduled, i.e. before each vehicle v departs, it is decided what time and what objects v would pass. In the railway systems, trains move according to the time table. In vehicle systems including large number and various kinds of vehicles in a complex space, it consumes much computation to make up the schedule. Hence, it is not easy to change the schedules if unexpected accidents occur. In the resolution method, each vehicle v requests to pass an object o any time v would like to pass o. If o is congested, v may autonomously change the way. In this paper, we adopt the resolution scheme where v locks o, i.e. *distributed* control. If the lock request is accepted, v can move to the objects.

The movement of each vehicle v on an object o is considered to be *atomic*, i.e. v can either pass o or not. Hence, we model the movement of v on o as a transaction which locks o before arriving at o and releases o after leaving o. Since o is composed of smaller subobjects, the movement on o is realized as a sequence of the movement on the subobjects. It is modeled as a *nested* transaction [10, 11]. There are multiple kinds of vehicles on how they can move on the objects. For example, airplanes cannot stop, and automobile cars on motorways cannot back in the inverse way on the motorways. For the vehicle which cannot back, there is no need to hold the objects passed by v. On the other hand, a vehicle v which can back may hold the objects which v has passed since v may back. Vehicles which cannot stop have to hold objects farther to them in order to make sure that they could move. In this paper, we discuss schemes for locking and releasing objects with respect to types of vehicles.

In section 2, we present a system model. In section 3, we discuss how to make a path for each vehicle. In section 4, we present a synchronization schemes by which vehicles lock objects as transactions. In section 5, we present the evaluation of the synchronization schemes.

2 System Model

We present a model of the vehicle system.

2.1 Vehicle space tree

A vehicle system **T** is composed of a collection **V** of vehicles and a space **S** where vehicles move. A set of roads is an example of the vehicle space where automobile cars move around as vehicles. **S** is partitioned into disjoint units named *objects*. A sequence of objects denotes a path which vehicles pass. Each object o is further partitioned into smaller objects o_1, ..., o_n. Here, o is a *parent* of o_i, and o_i is a *component* of o $(i = 1,...,n)$. Thus, **S** is structured in a tree named a *space tree* whose nodes denote objects and branches represent the parent-component relation among the objects. In **S**, let $lca(o_1, o_2)$ denote a *least common ancestor* (lca) of objects o_1 and o_2. For each o, let $level(o)$ be a level of o, i.e. number of nodes in a path from o to the root. o_1 and o_2 are at the *same* level $(o_1 \equiv o_2)$ iff $level(o_1) = level(o_2)$. o_1 is *higher* than o_2 $(o_1 \succ o_2)$ iff $level(o_1) > level(o_2)$. Higher-level objects denote *broader* paths than the lower-level objects. A tree is *balanced* iff every leaf in the tree is at the same level. In this paper, we assume that **S** is balanced.

Each object o has a set of ports $\{prt_1, ..., prt_h\}$ $(h \geq 1)$. Here, let $o{:}prt_j$ denote a port prt_j of o. A vehicle v can move from a port prt_j to prt_k in o. It is written as $prt_j{:}o{:}prt_k$, named a *primitive* path. $\langle o \rangle$ denotes some primitive path in o.

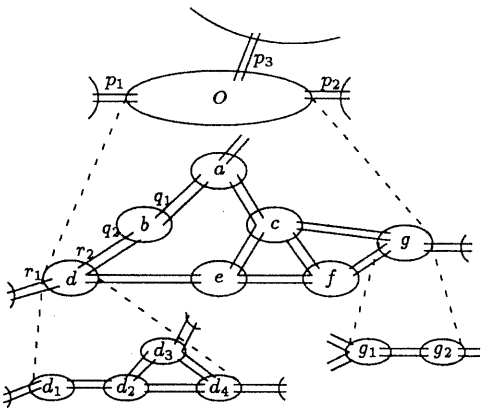

Figure 1: Space object

The same-level objects o_1 and o_2 are *directly connected* if there is a link between $o_1{:}prt_{1j}$ and $o_2{:}prt_{2k}$. If o_1 and o_2 are directly connected, vehicles can move between o_1 and o_2. In this paper, we assume that there exists at most one link between any two objects, and each port participates in only one link. Let $\langle o_1 \rangle$ and $\langle o_2 \rangle$ be primitive paths $prt_{1i}{:}o_1{:}prt_{1j}$ and $prt_{2k}{:}o_2{:}prt_{2l}$ o_1 and o_2, respectively. $\langle o_1 \rangle$ and $\langle o_2 \rangle$ are *directly connected* iff o_1 and o_2 are directly connected by a link between prt_{1j} and prt_{2k}. A *path* p is a sequence of primitive paths, i.e. $\langle \langle o_1 \rangle, ..., \langle o_n \rangle \rangle$ where every $\langle o_i \rangle$ and $\langle o_{i+1} \rangle$ are directly connected. Here, o_1 is the *source* and o_n is the *destination* of p,

[**Example 2.1**] Figure 1 shows that an object o is composed of disjoint component objects $a, b, \ldots, g$ which are further composed of smaller component objects. p_1, p_2, and p_3 are ports of o. $p_1 : o : p_2$, $p_1 : o : p_2$, and $p_2 : o : p_3$ are primitive paths. The double lines between objects indicate links. b and d are directly connected because there is a link between the ports g_2 of b and r_2 of d. There is at most one link between two objects. Figure 2 shows the space tree of Figure 1. o is the root. □

Each primitive path $\langle o \rangle$ is represented by a path $p = \langle\langle o_1 \rangle, \ldots, \langle o_n \rangle\rangle$ where each o_i is a component of o. p is written as $\langle o \rangle^1$. $\langle o \rangle^1$ is named a *component path* or *first expansion* of $\langle o \rangle$. The *i-th expansion* $\langle o \rangle^i$ is defined as $\langle \langle o_1 \rangle^{i-1}, \ldots, \langle o_n \rangle^{i-1} \rangle$ for $i \geq 1$, and $\langle o \rangle^0 = \langle o \rangle$. Since the space tree is finite, there exists a fixed point $\langle o \rangle^j$ such that $\langle o \rangle^j = \langle o \rangle^k$ for every $k \geq j$, and $\langle o \rangle^j \neq \langle o \rangle^k$ for every $k < j$. Let $\langle o \rangle^*$ denote the fixed point. There may be multiple candidates as $\langle o \rangle^i$. o selects an optimal one among the component paths so that the moving cost is the minimum. $\langle o_1 \rangle$ and $\langle o_2 \rangle$ are *indirectly connected* iff (1) *level*$(o_1) \neq$ say $o_1 \succ o_2$, and (2) for some j, $\langle o_2 \rangle$ and $\langle o_1 \rangle^j$ are directly connected. $\langle o_1 \rangle$ and $\langle o_2 \rangle$ are *connected* iff they are directly or indirectly connected. For example, $\langle o \rangle = p_1 : o : p_2$ can be expanded to multiple paths, e.g. $\langle\langle d \rangle, \langle e \rangle, \langle f \rangle, \langle g \rangle\rangle$, $\langle\langle d \rangle, \langle b \rangle, \langle a \rangle, \langle c \rangle, \langle g \rangle\rangle$ in Figure 1. o can select an optional one. $\langle\langle d_1 \rangle, \langle d_2 \rangle, \langle d_3 \rangle\rangle$ and $\langle b \rangle = q_1 : b : q_2$ are indirectly connected.

Let p_1 be a path $\langle\langle o_1 \rangle, \ldots, \langle o_n \rangle\rangle$. If a path p_2 is $\langle\langle o_1 \rangle, \ldots, \langle o_{i-1} \rangle, \langle o_i \rangle^j, \langle o_{i+1} \rangle, \ldots, \langle o_n \rangle\rangle$ for some j (> 1), p_1 is *broader* than p_2 (written as $p_1 \prec\!\!\prec p_2$). Further, $\prec\!\!\prec$ is transitive, i.e. if $p_1 \prec\!\!\prec p_3 \prec\!\!\prec p_2$, then $p_1 \prec\!\!\prec p_2$. The broader a path is, the higher-level objects are included in the path.

[**Definition**] Let p, q, and r be paths from an object s to d. q is a *broadest* path of p if $p \prec\!\!\prec q$ and there is no path r such that $q \prec\!\!\prec r$. q is a *most detailed* path of p if $q \prec\!\!\prec p$ and there is no r such that $r \prec\!\!\prec q$ □.

The most detailed path is composed of only leaf objects. For a path $p = \langle\langle o_1 \rangle, \ldots, \langle o_n \rangle\rangle$, if $o_1 \preceq \ldots \preceq o_n$, $o_1 \succeq \ldots \succeq o_n$, and $o_1 \equiv \ldots \equiv o_n$, p is an *ascent*, *descent*, and *flat* path, respectively. If every o_i is a component of some object o, p is *included in* o or is a *component* path of o.

Suppose that a vehicle v would like to move from an object s to d. Let o_s and o_d be component objects of $o = lca(s, d)$ which are ancestors of s and d, respectively. o_s and o_d are a *broadest source* and *destination* of v, respectively. A flat path from o_s to o_d, i.e. $\langle\langle o_s \rangle, \ldots, \langle o_d \rangle\rangle$ which includes only component objects of o is a *broadest* path from s to d, written as *broad*(s, d). For example, d and g are broadest source and destination of a vehicle which would move from d_1 to g_1 in Figure 1. $\langle\langle d \rangle, \langle e \rangle, \langle f \rangle, \langle g \rangle\rangle$ is an example of *broad*(s, d). Here, a current object of a vehicle v means an object where v is now.

2.2 Types of vehicles

There are types of vehicles on how they could move in the vehicle space. For example, airplanes cannot stop in the space, automobile cars in the motorways cannot back, packets in the network can be removed. The vehicles are classified with respect to the following three points.

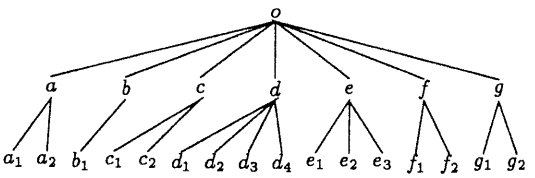

Figure 2: Space tree of Figure 1.

1. Vehicles are *removable* or not.
2. Vehicles are *stoppable* or not.
3. Vehicles are *retreatable* or not.

While packets in communication networks can be removed, airplanes cannot be removed. Problem is how to treat a vehicle v if v cannot move due to some events like unexpected accidents or deadlocks. For example, even if the vehicle system is deadlocked, no deadlocked vehicle like automobile cars can be removed although deadlocked transactions can be removed in database systems. In order to resolve the deadlock or make vehicles give up to hold objects, the vehicles have to move to another object from the current objects.

The second point is concerned with whether vehicles can stop or not. *Stopping* of a vehicle v at an object o means that v can stay in o as long as v would like to stay there. For example, airplanes cannot stop, but trains can stop on the railway. For each *unstoppable* vehicle v, objects farther from v have to be obtained to make sure that v pass the objects. On the other hand, objects farther from v need not be obtained by a *stoppable* vehicle v because v can wait in some objects if v could not obtain the objects.

The last point is concerned with whether or not v can back along the path passed by v. For example, packets in the networks can back, but automobile cars in motorways cannot back. A vehicle v which can back along the same path passed by v are a *retreatable* one. For each *unretreatable* vehicle v, as soon as v passes o, v can release o since v never backs. On the other hand, if v is retreatable, v can hold o since v may back the same path which v has passed. Here, v on o_1 *retreats* to some o_2 iff v has taken some path p from o_2 to o_1 and takes the same objects in p in the inverse way. v on o_1 *backs* to o_2 iff v moves from o_1 to o_2 whether or not v takes the same path as v has taken.

2.3 Properties of objects

For each primitive path $\langle o \rangle$, the *capacity* of $\langle o \rangle$, $cap(\langle o \rangle)$ represents how many vehicles $\langle o \rangle$ can admit at the same time. The *moving time* of $\langle o \rangle$, $time(\langle o \rangle)$ means how long it takes each vehicle to pass $\langle o \rangle$ if there is no vehicle in $\langle o \rangle$. For a path $p = \langle\langle o_1 \rangle, \ldots, \langle o_n \rangle\rangle$, $cap(p) = cap(\langle o_1 \rangle) + \ldots + cap(\langle o_n \rangle)$ and $time(p) = time(\langle o_1 \rangle) + \ldots + time(\langle o_n \rangle)$. In this paper, we assume that each vehicle cannot manage its speed. For two primitive paths p_1 and p_2 in o, $p_1 \cup p_2$ means a collection of p_1 and p_2. p_1 and p_2 are *independent* iff $cap(p_1 \cup p_2) = cap(p_1) + cap(p_2)$. In this paper, we assume that every primitive paths are indepen-

dent. The *throughput* of $\langle o \rangle$, $thru(\langle o \rangle)$ is $cap(\langle o \rangle)$ / $time(\langle o \rangle)$. The larger $thru(\langle o \rangle)$ gets, the more vehicles can pass $\langle o \rangle$.

Let $\langle a \rangle$ and $\langle b \rangle$ be primitive paths directly connected with a primitive path $\langle o \rangle$, $\langle\langle a \rangle, \langle o \rangle, \langle b \rangle\rangle$. Let PA be a set of component paths of $\langle o \rangle$ indirectly connected with $\langle a \rangle$ and $\langle b \rangle$. A path p in PA is the *most significant* for $\langle o \rangle$ iff $cap(p)$ is the maximum in PA. The most significant path represents a *trunk* path like the motorway. For example, if the motorway is congested, the roads around the motorway are congested. Also, vehicles take the motorway if there exists. Thus, the trunk path represents all the paths in PA.

Suppose that there are component paths $p_1, \ldots, p_h$ for a primitive path $\langle o \rangle$. $cap(\langle o \rangle)$ and $time(\langle o \rangle)$ are computed from the components as follows. Let p_i be the most significant component path of $\langle o \rangle$. Here, $cap(\langle o \rangle) = cap(p_i)$, and $time(\langle o \rangle) = time(p_i)$.

$hold(\langle o \rangle)$ denotes a number of vehicles which are now in $\langle o \rangle$ ($hold(\langle o \rangle) \leq cap(\langle o \rangle)$). $cong(\langle o \rangle)$ is a *congestion* factor, i.e. $hold(\langle o \rangle)$ / $cap(\langle o \rangle)$. $ptime(\langle o \rangle)$ is time when it takes a vehicle to pass $\langle o \rangle$. It is computed as $time(\langle o \rangle)$ / (1 - $cong(\langle o \rangle)$). The more congested $\langle o \rangle$ is, the longer time it takes to pass $\langle o \rangle$. If $cong(\langle o \rangle) = 1$, i.e. $\langle o \rangle$ is fully congested. $ptime(\langle o \rangle)$ is infinite. o selects $\langle o \rangle$ with the minimum $ptime(\langle o \rangle)$. It is not easy to get from every component information on how much each component path is congested. Hence, the congestion factor of the most significant primitive path of $\langle o \rangle$ is used to represent how much $\langle o \rangle$ is congested. Every component informs the parent of the congestion factor of the most significant primitive path periodically or when the state is changed, e.g. some accident occurs. Based on the information from the components, the object decides a component path for a primitive path.

3 Path Decision Strategy

We would like to discuss how to decide a path to the destination for a vehicle v. Here, suppose that v would like to move from an object s to d.

1. First, the broadest source o_s and destination o_d of v are found for s and d, respectively. $o = lca(s, d)$ makes a broadest path $broad(s, d)$ from o_s to o_d. It is rewritten as $p = \langle\langle o_1 \rangle, \ldots, \langle p_n \rangle\rangle$.
2. p is more detailed as $\langle\langle o_1 \rangle^{d_1}, \ldots, \langle o_n \rangle^{d_n}\rangle$ where $d_j \geq 0$ and $d_i \geq d_j$ for $i < j$.

$\langle o_i \rangle$ is more detailed than $\langle o_j \rangle$ if $i > j$, i.e. o_i is nearer to v than o_j. d_j is computed as follows. Here, let h denote the height of the space tree minus $level(o)$, i.e. every j, $\langle o_j \rangle^k = \langle o_j \rangle^*$ for every $k \leq h$. Here, $d_1 = h$, i.e. $\langle o_1 \rangle$ is the most detailed. Figure 3 shows d_j for j.

[Expansion level]

1. $d_j = h$ for $j \leq I$.
2. $d_j = h \times (1 - ((j - I)/n)^2)$ for $I < j \leq H$.
3. $d_j = h/j$ for $j > H$. □

I and H are threshold parameters. The objects inside H from the current object are sufficiently detailed for vehicles to surely pass through them. The objects outside H are broader. For unstoppable vehicles, I is bigger than stoppable ones in order to make sure that the vehicles can go ahead.

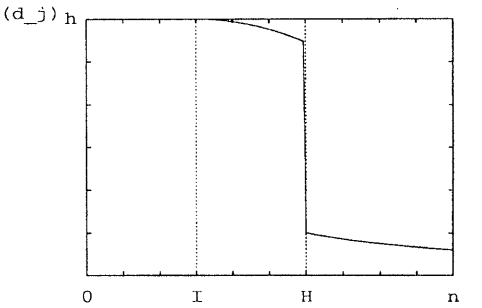

Figure 3: d_j

4 Synchronization

Each movement of a vehicle v in an object o is *atomic*, i.e. v can either pass o or not. We model the movement of v as a transaction [4].

4.1 Locking scheme

Even if a path p for v is decided by the path decision strategy, v cannot pass an object o in p if o is fully held by other vehicles. In order to make sure that v can pass o, v locks o before arriving at o. If v locks o, v is assured to be able to pass o. After leaving o, v can release o. If $cap(\langle o \rangle) - hold(\langle o \rangle) \geq 0$, o can accept further lock requests from vehicles. Otherwise, vehicles which would like to lock o cannot hold o.

Suppose that v in o_1 would like to pass $\langle o \rangle$. First, v sends a lock request to o.

[Locking scheme]

(1) If $cap(\langle o \rangle) - hold(\langle o \rangle) \geq 0$, o accepts the lock request from v. Then, o requests locks on all the ancestors of o for v. If all of the ancestors are cannot be locked, v cannot hold o. If locked, $hold(\langle o \rangle) = hold(\langle o \rangle) - 1$.

(2) Otherwise, o rejects the lock request from v. That is, v cannot hold o. □

The lock request to o is propagated to the ancestor objects of o. If all the ancestors could be locked, o can be locked. After v locks $\langle o \rangle$, v requires o to decide a component path $\langle\langle o_1 \rangle, \ldots, \langle o_n \rangle\rangle$ of $\langle o \rangle$. v is a sequence of subtransactions $v_1, \ldots, v_n$, where each v_j is concerned with an atomic movement on $\langle o_j \rangle$ ($i = 1, \ldots, n$). v_j is further realized by a sequence of subtransactions $v_{j1}, \ldots, v_{jm_j}$ on the components $o_{j1}, \ldots, o_{jm_j}$ of o_j. If v passes through o_j, the subtransaction v_j commits. If all the subtransactions commit, v commits. Unless v can pass o_j, i.e. v cannot lock $\langle o_j \rangle$, v_j aborts. In the conventional database systems, a transaction aborts, i.e. whole update effect done by the transaction is removed. On the other hand, only the part of the transaction can be aborted, i.e. *partial abortion* [12, 17, 18]. That is, another path may be tried to be found from o_{j-1} if v aborts. By backing to some o_k ($k < j$), another path for o_k may be tried to be found. o tries to find alternate paths for o_{j-1} It can

not be seen from the higher-level of o, i.e. the movement on o is atomic. Thus, the vehicle transaction v on o is *nested* [10, 11, 16].

4.2 Releasing schemes

After leaving o, v can release o. If v is a *strict two - phase locked* (*2PL*) [3] transaction, v does not release objects until v arrives at the destination. This means that objects which v has passed through already cannot be used by another vehicle. It decreases the number of vehicles which are moving at the same time. On the other hand, if v releases objects which v has passed, the objects can be used by another vehicle. It implies that more vehicles can move at the same time in the space. We would like to consider how v releases locks on primitive paths which v has obtained. There are three ways to release objects. Here, suppose that $\langle o \rangle^1 = \langle \langle o_1 \rangle, \ldots, \langle o_n \rangle \rangle$ and v_i denotes a subtransaction of v on o_i.

[Schemes for releasing objects]

1. If v is a root of the vehicle, i.e. v arrives at the destination, all the locks held by v are released. Otherwise, no objects are released.
2. $\langle o_1 \rangle, \ldots, \langle o_n \rangle$ held by the subtransactions $v_1, \ldots, v_n$ of v are released.
3. $\langle o \rangle$ is released, and all the paths obtained by $v_1, \ldots, v_n$ are released if they are still obtained. □

The first scheme is a *close* scheme which is a strict *2PL* [3]. That is, only when the whole transaction commits, i.e. the vehicle arrives at the destination, all the locks obtained are released. The second scheme is a *semi-open* one. It is noted that $\langle o \rangle$ is still held by v while $\langle o_1 \rangle, \ldots, \langle o_n \rangle$ obtained by $v_1, \ldots, v_n$ of v are released. When v passes o, v releases all the primitive paths included in the component path of $\langle o \rangle$ but does not release $\langle o \rangle$. The third scheme is an *open* one where all the objects obtained by v are released. This means that v releases $\langle o \rangle$ as soon as v passes o.

4.3 Relation among vehicle types and releasing schemes

It depends on the vehicle type of v which scheme is adopted to release objects. Suppose that v is *unretreatable*, i.e. v never backs along the path passed by v. Hence, as soon as v passes objects, v can release them, i.e. unretreatable vehicles can adopt the open scheme. A retreatable vehicle v may back to some object which v has passed. For example, if v finds that v cannot go ahead according to the path due to the congestion and deadlock, v may back along the path which v has passed. In this case, if the objects which v has passed are released, v cannot back. Therefore, v can adopt the semi-open or close scheme. If the close scheme is adopted, v can retreat, i.e. v can reverse the same path taken by v. In the semi-open scheme, v can find another path to back by detailing component paths of higher-level objects which v still holds. Since the semi-open scheme holds less objects than the close one, the semi-open implies that more vehicles can be admitted in the space than the close one.

If o cannot accept v, e.g. o is congested, v wants on o or tries to find another way. The first is a *wait* action. In the database systems, transactions wait until they could get the objects. As stated before, although the stoppable vehicles can take the wait action, the unstoppable vehicles like airplanes have to go ahead without stopping. The second is a *re-planning* action. If o is outside H, i.e. o is farther from the current object, v may carry on to take the path. If o is inside H, v tries to find another path to escape o. In addition to the conventional wait, the re-planing is adopted in our system. Suppose that v is now in o_i and tries to move to o_j adjacent to o_i. If v can lock o_j, v leaves o_i for o_j and then releases o_i. When v adopts the replanning strategy, the path for v has to be canceled by releasing the objects. Then, a new path is decided.

Table 1 summarizes the relation between the vehicle types and the releasing schemes.

5 Evaluation

We evaluate the close, semi-open, and open schemes in terms of the number of objects held by the vehicle. We assume that a space tree T has a height and breadth balanced tree whose height is l and where each non-leaf object has k subobjects. The lowest-level path includes k^l lowest-level objects. Assume that a vehicle v moves on all the objects from the left-most lowest-level object to the right-most one, i.e. v passes k^l lowest-level objects. Suppose that v is in a lowest-level object o_l in T. A parent of o_l is denoted by o_{l-1}. Thus, o_i is an ancestor of o which is at a level $i(\leq l)$. Here, o_0 denotes the root of T. o_1 is the a_1-*th* component object of the root, i.e. o_o. o_i is the a_i-*th* component object of o_{i-1} $(i = 1, \ldots, k)$. The position of v in T is represented by $\langle a_1, \ldots, a_l \rangle$, where each $a_i \leq k$.

In the open scheme, v holds only objects o_l, o_{l-1}, $\ldots$, o_1, o_0 since v releases the objects as soon as v leaves them. Hence, v on o_l holds $(l+1)$ objects. In the semi-open scheme, v on o_l holds $(1+a_1+\ldots+a_l)$ objects since v does not release the ancestor object. In the close scheme, v on o_l holds $(1+(a_1-1)(1+k+\ldots+k^{l-1}))+(1+(a_2-1)(1+k+\ldots+k^{l-2}))+\ldots+(1+(a_{l-1}-1)(1+k))+(1+(a_l-1))+1 = (l+1)+(a_1-1)(1-k^l)/(1-k)+(a_2-1)(1-k^{l-1})/(1-k)+\ldots+(a_{l-1}-1)(1-k^2)/(1-k)+(a_l-1)$ since v does not release the objects. Let N_O, N_{SO}, and N_C be the average numbers of objects held by v in the open, semi-open, and close scheme, respectively. $N_O = l+1$, $N_{SO} = ((l+2)/2)+(kl/2)$, $N_C = ((l+2)/2)+l/2$ (for $k=1$), $((l+2)/2)+(k/2)(k^l-1)/(k-1)$ (for $k>1$) Figure 4 shows N_O, N_{SO}, and N_C for k given $l = 10$. The semi-open transaction holds less objects than the close one. The semi-open scheme locks $O(k)$ objects while the close locks $O(k^l)$ objects.

6 Concluding Remarks

In this paper, we have discussed how to model the movement of a vehicle in a tree-structured space. The vehicle movement is modeled as a nested transaction in the vehicle space. The vehicles are classified with respect to three points, *removable* or not, *stoppable* or not, and *retreatable* or not. While transactions in the database systems can be removed by aborting the transactions, most vehicles like automobile cars cannot be removed. The unremovable vehicle has to move

Table 1: Releasing schemes and vehicle types

Releasing schemes	Concurrency degree	Retreatable degree	Vehicle types	
			Retreatable	Nonretreatable
close	×	○	○	×
semi-open	△	△	○	×
open	○	×	×	○

○ : good. △ : marginal. × : not good.

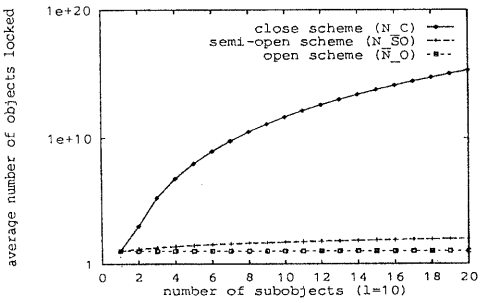

Figure 4: Evaluation

to another object in order to release the objects which it holds. We have shown three schemes for releasing objects, i.e. *open*, *semi-open*, and *close* ones. The close scheme corresponds to the conventional strict two-phase locking one where the objects are released when the vehicle arrives at the destination. In the open and semi-open schemes, the objects are released before the vehicle arrives at the destination. We have discussed how each kind of vehicle can adopt a synchronization scheme. We have shown the evaluation of the releasing schemes in terms of the number of objects locked.

References

[1] Bernstein, P. A., Hadzilacos, V., and Goodman, N., "Concurrency Control and Recovery in Database Systems," *Addison Wesley*, 1987.

[2] Deen, S. M., Hamada, S., and Takizawa, M., "Broad Path Decision in Vehicle System," *Proc. of the 3rd Int'l Conf. on Database and Expert Systems Applications (DEXA'92)*, 1992, pp.8-13.

[3] Eswaren, K. P., Gray, J., Lorie, R. A., and Traiger, I. L., "The Notion of Consistency and Predicate Locks in Database Systems," *CACM*, Vol.19, No.11, 1976, pp.624-637.

[4] Gray, J., "The Transaction Concept: Virtues and Limitations," *Proc. of the 7th VLDB*, 1981, pp.144-154.

[5] Hamada, S. and Takizawa, M., "Transaction Model of Automated Guided Vehicles," *Preprints of the JSPE-IFIP WG5.3 Workshop on the Design of Information Infrastructure Systems for Manufacturing DIISM'93*, 1993, pp.281-292.

[6] Holt, R. C., "Some Deadlock Properties on Computer Systems," *ACM Computing Surveys*, Vol.14, No.3, 1972, pp.179-196.

[7] Knapp, E., "Deadlock Detection in Distributed Databases," *ACM Computing Surveys*, Vol.19, No.4, 1987, pp.303-328.

[8] Korth, H. F., "Locking Primitives in a Database System," *JACM*, Vol.30, No.1, 1983, pp.55-79.

[9] Korth, H. F., Levy, E., and Silberschalz, A., "A Formal Approach to Recovery by Compensating transactions," *Proc. of the VLDB*, 1990, pp.95-106.

[10] Lynch, N. and Merritt, M., "Introduction to the Theory of Nested Transactions," *MIT/LCS/TR 367*, 1986.

[11] Moss, J. E., "Nested Transactions: An Approach to Reliable Distributed Computing," *The MIT Press Series in Information Systems*, 1985.

[12] Takizawa, M. and Deen, S. M., "Lock Mode Based Resolution of Uncompensatable Deadlock," *Proc. of the Far-east Workshop on Future Database Systems*, 1992, pp.168-175.

[13] Takizawa, M., Hamada, S., and Deen, S. M., "Vehicle Transactions," *Proc. of the 4rd Int'l Conf. on Database and Expert Systems Applications (DEXA'93)*, 1993, pp.611-614.

[14] Traiger, I. L., "Trends in System Aspects of Database Management," *Proc. of the 2nd Int'l Conf. on Database (ICOD-2)*, 1983, pp.1-21.

[15] Weihl, W. E., "Local Atomicity Properties: Modular Concurrency Control for Abstract Data Types," *ACM Trans. on Programming Language and Systems*, Vol.11, No.2, 1989, pp.249-283.

[16] Weikum, G., "Principles and Realization Strategies of Multilevel Transaction Management," *ACM TODS*, Vol. 16, No. 1, 1991, pp.132-180.

[17] Yasuzawa, S., Takizawa, M., and Ouchi, T., "Resolution of Parallel Deadlock by Partial Abortion," *Proc. of the International Symposium on Communications (ISCOM)*, 1991, pp.708-711.

[18] Yasuzawa, S. and Takizawa, M., "Uncompensatable Deadlock in Distributed Object-Oriented Systems," *Proc. of the International Conference on Parallel and Distributed Systems (ICPADS)*, 1992, pp.150-157.