A Two-Sided, High-Density Rail-Rail Hub

Gang Hao and Kevin R. Gue
University of Louisville

July 8, 2016

Abstract: *We describe a control scheme for a two-sided, grid-based Rail-Rail Physical Internet hub. The system features decentralized control, and is able to induct and discharge multiple π-containers at the same time. It also sorts and sequences containers, such that inducted containers are placed in the right slot, in the rail car, at the right time.*

Keywords: *GridStore, Physical Internet, high-density storage system, grid-based, GridHub, Rail-Rail hub*

1 Introduction

The Physical Internet (PI or π) was introduced as a next generation logistics system by Montreuil (2011). Its objective is to improve sustainability and efficiency of logistics systems by standardizing containers' sizes, transportation, and means of material handling. Hubs in a PI network (π-hubs) serve functions similar to routers in the Digital Internet—π-containers (data packages) are received via some dock doors (ports) of π-hubs, and are delivered to specific departure locations in specific sequences.

In this paper, we present a two-sided rail-to-rail (Rail-Rail) container hub and describe a grid-based π-hub system which is able to receive, temporarily store, and accurately distribute unit-sized π-containers. The system is developed according to the conceptual design of Eric Ballot and Benoit Montreuil and Collin Thivierge (2012). Our particular implementation features decentralized control of a grid of unit-sized conveyors. The control algorithm extends the grid-based control algorithms of Gue et al. (2012), Gue et al. (2014), and Uludag (2014).

2 Literature Review

Conceptual designs of π-hubs were addressed after the introduction of The Physical Internet. The design of a road-road hub was done by Meller et al. (2013). Eric Ballot and Benoit Montreuil and Collin Thivierge (2012) proposed a design for a road-rail hub. The authors of these papers described the material flows of the hubs in conceptual terms, without giving detailed algorithms for the movement of containers in a grid-based storage system.

Pach et al. (2014) proposed a potential field approach for routing π-containers in a rail-road π-hub, and Sallez et al. (2015) gave a possible control architecture for the routing problems of π-containers in a cross-docking π-hub. Their works were based on the π-hub structures that were described in the conceptual designs (Meller et al., 2013; Eric Ballot and Benoit Montreuil and Collin Thivierge, 2012), and both studies also suggested interactions among π-conveyors in order to move π-containers. Detailed algorithms were also lacking in these papers.

Gue et al. (2014) were the first to describe a detailed control algorithm (GridStore) for large-scale, grid-based storage and retrieval systems. The GridPick (Uludag, 2014) and the GridSequence (Gue et al., 2012) algorithms extended the GridStore algorithm for different applications. Each of these methods enables a grid of square conveyors to move unit-sized boxes under decentralized control.

A competing method for controlling a grid-based system is the path-reserve approach, which was first introduced by Mayer (2009) for the Flexconveyor system. This method has been extended to a grid-based sorting system (GridSorter) by Seibold et al. (2013). GridSorter transfers containers to fixed destinations, but not necessarily in a required sequence.

The algorithm we propose is different than existing literature in a number of ways:

- Our algorithm modifies existing GridStore and GridPick methods such that north-south and east-west movement is accomplished simultaneously.
- Containers leave our system in a predetermined sequence, which is a feature not offered by the GridSorter system.
- We apply our methods to a rail-rail π-hub, and show how the system responds to different levels of load in simulated operations.

3 The Two-Sided GridHub

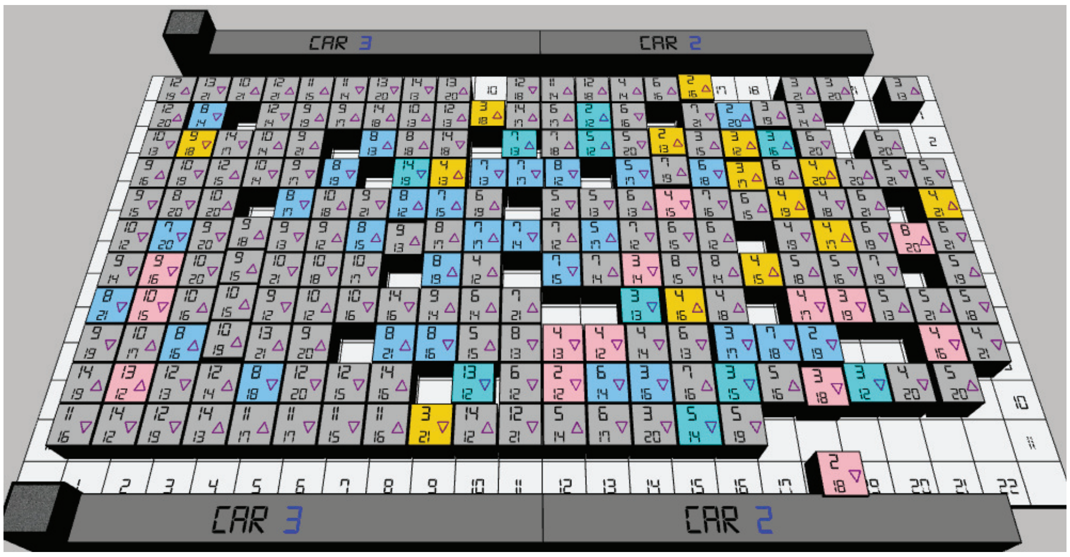

Figure 1: GridHub overview

Figure 1 illustrates a two-sided, grid-based rail-rail hub. The two longer edges are used as loading faces to accept or distribute π-containers. Two trains can stop along the loading faces to load and unload containers at the same time. The loading faces are slightly longer than two railway cars; therefore, railway cars can stop to unload containers on the left side first, then move forward "one step" to the right to load containers. Conveyors at the ends of every column are numbered in Figure 1. These conveyors serve as gates or departure locations used to load or unload containers. In the illustration, locations numbered 2 to 11 are used to receive π-containers, and conveyor 12 to 21 are used to release containers for both of the loading faces. The components of this system are as follows:

A group of conveyors In Figure 1, one blank square represents a conveyor. Each of the conveyors can carry only one π-container, and it can move the π-container placed on it to one of its four cardinal neighbors.

A central controller This controller can be considered as a counter plus a train dispatcher. The counter counts the number of arrived railway cars, and its value can be obtained by every conveyor. The dispatcher is able to ask the train to move or stop according to the counter's state. They work together to keep the system running.

Railway cars In Figure 1, railway cars are shown as long rectangular boxes. The space in every railway car is divided into several slots to hold π-containers, and one slot holds only one π-container. When the train is stopping along the loading face, each of the slots directly faces a gate.

A group of π-containers In Figure 1, one gray cube represents a π-container. Departure information is displayed on each container for illustration purposes. This information includes three items (Figure 2):

- Railway car of departure, indicating which railway car a container will be loaded into. This information also determines the sequence of departure. In Figure 2, the numbers on the upper part of every container show this information.
- Location or gate of departure, which determines the slot in a railway car that a container will be loaded into. Because the slot on the railway car directly faces a gate, this information also indicates through which column of conveyors the container will leave the hub. In Figure 2, the numbers on the lower part of every container show this information.
- Edge of departure, showing the edge of the hub on which a container will be released. In Figure 2, the small triangle on every container indicates the edge of departure.

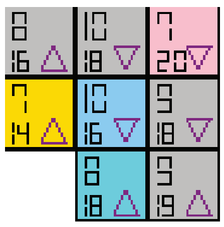

Figure 2: Departure information of the π-containers

We set additional assumptions:

- All containers and conveyors are of unit-size.
- Each railway car is fully loaded and unloaded, and each holds the same number of π-containers.
- A π-container's departure information is unchanged while in the system.
- Railway cars on both sides enter and leave the system at the same time. For instance, in Figure 1, if one of the cars numbered '2" completes loading activities, it will wait for the other car numbered '2".
- The quantity of π-containers in the system is constant.

3.1 Operation Description

A flow chart in Figure 3 illustrates the work flow of this π-hub.

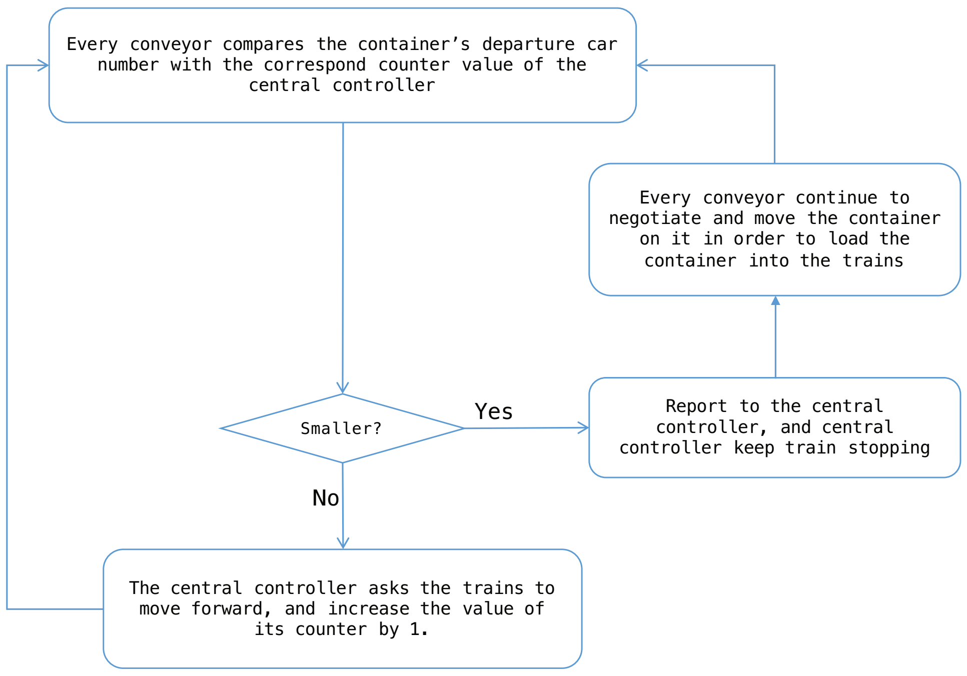
Every conveyor compares the container's departure car number with the correspond counter value of the central controller
Every conveyor continue to negotiate and move the container on it in order to load the container into the trains
Smaller?
Yes
Report to the central controller, and central controller keep train stopping
No
The central controller asks the trains to move forward, and increase the value of its counter by 1.

Figure 3: Operation flow

3.2 Control Algorithm

The control algorithm is an extension of existing grid-based control algorithms (Gue et al., 2012, 2014; Uludag, 2014). The algorithm takes the containers' departure information and system states as input, and the result is that every container is delivered to its specific departure location in a particular order.

3.2.1 States of Conveyors

Definitions of conveyor states are the same as developed by Gue et al. (2014).

- Active: the resident container must be transferred to a certain location. If a conveyor is in this state, it knows the direction to move the container and the target location. The colors of containers are used to represent the requested direction. In Figure 2: blue$\rightarrow$right; cyan$\rightarrow$left; gold$\rightarrow$up; pink$\rightarrow$down. For example, in Figure 2 a conveyor knows to move the container to the right, and the target location is the 16th column of conveyors.

- Occupied: the resident container is not currently requested. If a conveyor is in this state, it only holds the container's departure information.
- Empty: the conveyor is empty.

The information associated with a container is called its *Container Information Package (CIP)*. This is a data package held by conveyors, which includes a container's departure information and task information, such as target location and transfer direction. The departure information cannot be changed, but the task information can be changed by conveyors.

3.2.2 Difficulties

Due to the settings and requirements of this two-sided π-hub, some difficulties must be overcome to move containers to their target locations.

Fully occupied rows and columns An entire row or column of conveyors is occupied, and no space is left for the containers moving perpendicular to that row or column. The other case is similar: though the column or row of conveyors is not fully occupied, it is impossible for the active conveyor to find space for future transfers (this case can be called "half fully occupied" row or column). Examples of these two cases are in Figure 4.

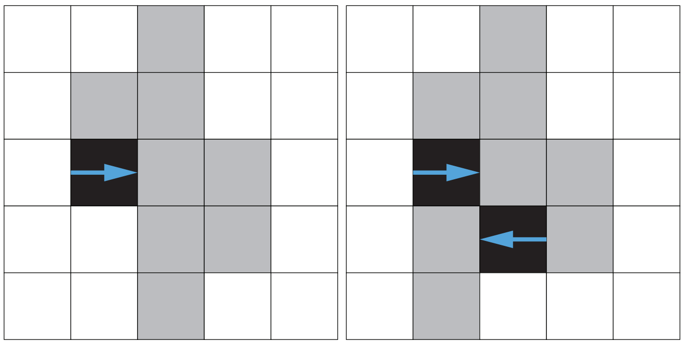

Figure 4: Example of fully (left) and half fully (right) occupied column or row

Move one or multiple containers among a group of conveyors repetitively In this case, one or more containers are being moved cyclically, and these actions block other movements. For example, in Figure 5, the marked container is moved between the two conveyors.

Resource conflicts for active conveyors There are several cases for this type of problem. Active conveyors with different requested directions compete for one empty conveyor or for an occupied conveyor which can make empty space. Some examples are shown in Figure 6.

3.2.3 Algorithm Description

The algorithm proceeds in phases, and the combination of these phases is called one iteration or one cycle of the algorithm. All conveyors run one iteration simultaneously and repetitively. The number

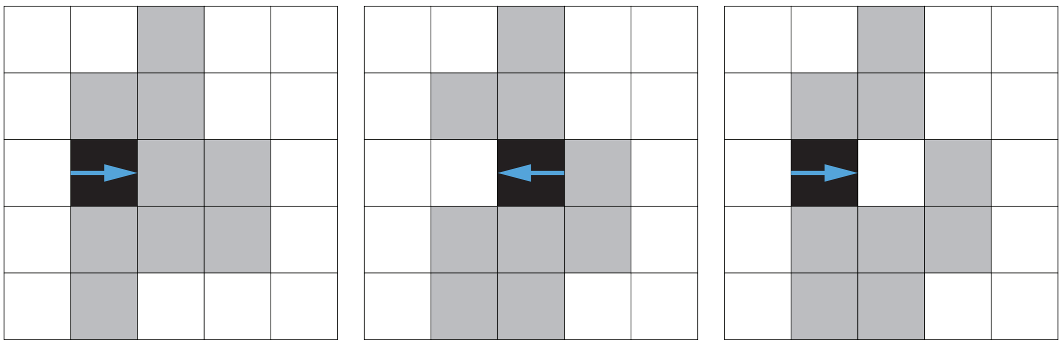
Figure 5: Example of cyclically movements

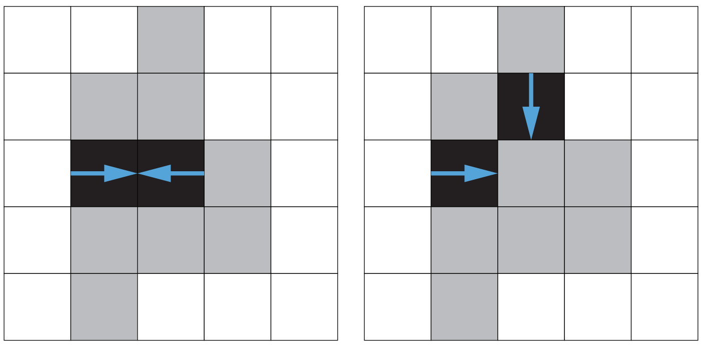
Figure 6: Examples of conflicts between active conveyors

of iterations or cycles performed can also be used as a measurement of time taken to completed some actions. For example, it takes four cycles to move one container out of the system.

Assess phase In this phase, the CIP held by every active conveyor is evaluated by the active conveyor. If the CIP shows that the current conveyor is the target location of the container, the conveyor changes its state from active to occupied. Otherwise, the conveyor stays active and seeks space to move the container in future negotiations. Occupied and empty conveyors do not need to perform any actions in this phase.

Negotiation phase

1. This negotiation detects fully occupied columns or rows of conveyors. Compared to the "balancing" actions of the GridPick algorithm (Uludag, 2014), negotiations to make space for requested items' movements only take place when entire rows or columns of conveyors are occupied. The first step is to make every conveyor detect the fully occupied cases. Detection is initiated by conveyors at one end of every column or row, and they initialize messages toward the other ends. Messages are passed along columns or rows of conveyors, and they are only passed by the occupied or active conveyors. Hence, if the messages reach the other ends, the conveyors which passed them are in fully occupied columns or rows. Conveyors on the other ends send reply messages to give each member of the columns or rows a mark to indicate the fully occupied cases.
2. Every conveyor reads the counter values of the central controller. If the departure car's value

in the CIP is smaller than the central counter's value (this means the railway car is waiting for containers):

- If the current column is not the same as the CIP's departure location, the conveyor changes to active state with left or right direction first.
- Otherwise, it changes state from occupied to active with requested direction toward the CIP's departure edge.

3. In a similar way, to detect fully occupied rows or columns, active conveyors not located in fully occupied rows or columns initialize messages to detect the "half fully occupied" cases after assigning transfer tasks.

4. Every active conveyor initializes messages to "break" the full rows or columns perpendicular with their moving direction. The idea of this negotiation is similar to the "balancing" negotiations in the GridPick system (Uludag, 2014), but it only takes effect on full or half full cases. Two methods are used to make spaces.

   - First, every active conveyor tries to "pull" one container back (opposing its requested direction) from the column or row in front of it or within the column or row in which it is located. An example is shown in Figure 7, and the problem shown in Figure 4 can be solved by this method.

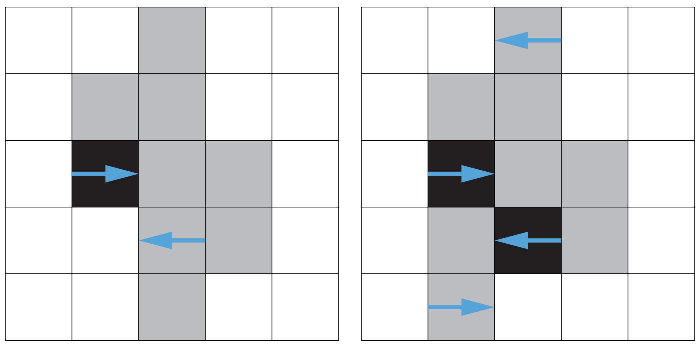

Figure 7: Example of "break" the fully and half fully occupied columns or rows (method 1)

   - Second, if the first method fails, some of the containers are "forced" to move out from the fully occupied rows or columns according to CIP and the central controller's information. An example is shown in Figure 8: if the first method fails to solve the case shown in the first picture of Figure 4, then the second method can solve it. The targets of these forced moving containers can be set to the columns which their departure gates locate or their neighbor conveyors.

   During this step, conflicts may occur. To solve the conflicts, conveyors with right/left active directions are defined always to have higher priority than the ones with up/down directions. Hence, negotiations to break full columns always take place prior to negotiations to break full rows. If two conveyors are competing to move one container towards their opposite directions, the conveyor holding the chosen container makes the decision on which direction to be pulled back randomly.

5. Every conveyor on the loading faces starts negotiation to move the container into the system if it holds a newly arrived container. The idea and effects of this negotiation are similar to "exchange of home row" negotiation in the GridStore system (Gue et al., 2014).

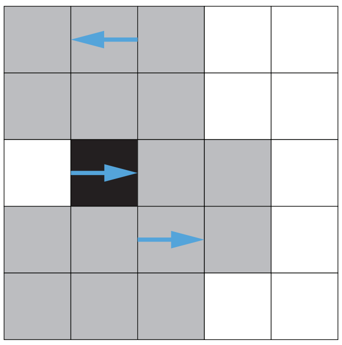

Figure 8: Example of "break" the fully occupied columns or rows (method 2)

6. Active conveyors initialize messages to eliminate the "face to face conflicts." Because the hub moves containers in four directions, a completely fixed priority scheme could cause some areas in the system to become crowded. The right/left directions are still set to have higher priorities, but every conveyor is set to choose priority between the pair of left-right (or up-down) in every cycle randomly to remove conflicts and to keep containers distributed evenly. For example, in the first picture of Figure 9, the left requested conveyor chooses left as the priority direction randomly. Due to its being requested to the left, it then sends a cancellation message to its left neighbor, but otherwise it keeps silent. If its left neighbor is an active conveyor with right direction, then its neighbor changes the active state to occupied by receiving the cancellation message. In the next step negotiation, its neighbor is moved away to make space for it, as shown in the second picture.

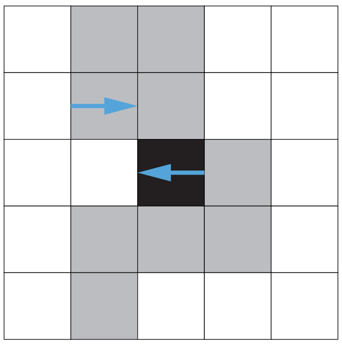

Figure 9: Example of solving conflict of two active conveyors with opposite directions

7. Negotiations for movements. In the same way described above, active conveyors with left/right direction have higher priority than the ones with up/down direction. Hence, higher priority ones start negotiations for movement before the lower ones start, and in the first wave of negotiations, the lower priority ones are considered as occupied instead of active. As a comparison, the negotiations initialized by the up/down active conveyors cannot take left/right active ones as the regular occupied ones. After the negotiations are done, every conveyor with a ready to be moved container has a confirmation mark indicating where to move the container, and the system enters the convey phase.

Convey phase In this phase, every conveyor with confirmation moves the container physically as the confirmation mark indicated. While every conveyor is moving its container, it also resets itself, and then sends the CIP to the destination conveyor. To prevent cyclical moving (explained in 3.2.2), it also sends a message to “tell” the destination not to assign a task to move the container back in a few iterations.

4 Performance and Discussion

We investigate the performance of the system with two parameters of system settings:

- The quantity of containers kept in system, and
- The number of rows of conveyors.

The performance metric is the number of railway cars processed during the run.

4.1 Experiment Settings and Results

We set the number of columns of conveyors to 22, and each of the railway cars can carry 10 π-containers. The two factors' settings are shown in Table 1.

Table 1: Experiment settings

Number of containers	Number	of	rows
120	11	12	13
140	11	12	13
160	11	12	13
180	11	12	13
200	11	12	13

The algorithm is programmed in Anylogic, and every configuration in Table 1 is set to run 6800 iterations and replicate 50 times. The 6800 iterations includes the first 800 iterations as warm-up period. Performance is shown in Figure 10.

4.2 Discussions

Both of the factors change the system storage density. Performance decreases with increasing in storage density. If we consider the system as a server in a queueing system, its performance is determined by its service time, which is the interval between loading two adjacent railway cars. The average values of intervals of each configurations in Table 1 are displayed in Figure 11. As expected, the interval increases with increasing storage density, and the longer intervals result in fewer cars processed.

Storage density has different mechanisms to affect the system performance at different levels.

Low storage density When the system density is low, the speed of moving requested items out of the system is faster (Gue et al., 2014), but retrieval time for every container is longer than the cases of higher density. This makes the speed not increase rapidly with decreasing of storage density.

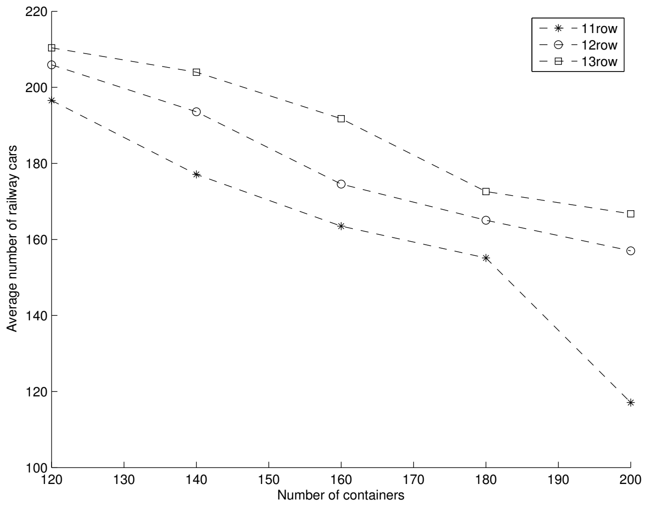

Figure 10: Average number of railway cars processed

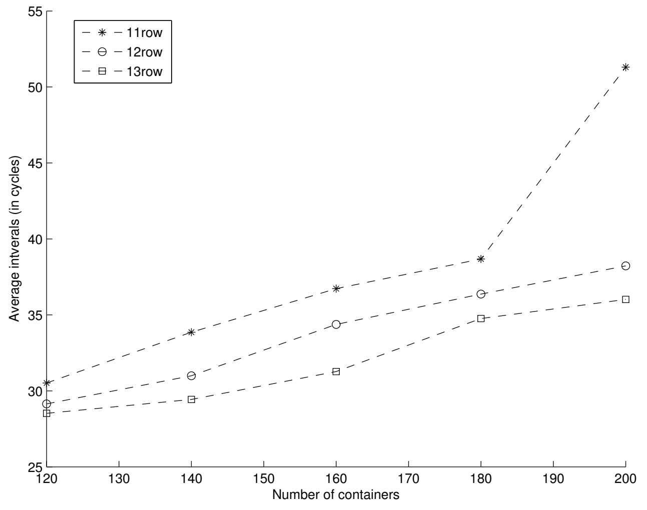

Figure 11: Average interval length

Retrieval time of a container in this paper is defined as the number of iterations taken to move out of the hub, and it is counted from the iteration during which the container's target railway car arrives at the loading position. Average retrieval times are shown in Figure 12. Containers can be moved to their target locations by requests generated from the second method described in 3.2.3. In low density cases, the frequency of these negotiations taking place is also low, and this causes most of the containers to be far away from their departure locations when the system starts loading them. Hence, when system density is low, for a single container, the average time to retrieve it is longer than the cases of higher density. However, the lower storage densities made loading faster, resulting in shorter loading intervals.

High storage density When the system's density is high, more negotiations (the second method described in 3.2.3) are performed. This results in more containers being transferred to places that are close to their departure locations prior to their target trains arriving, and the retrieval time of a single container gets shorter. However, intensive negotiations (the first method described in 3.2.3) delay the overall retrieval time.

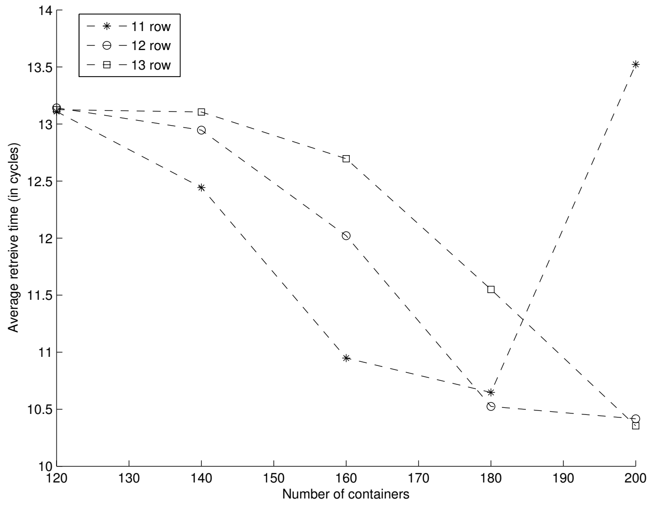

Figure 12: Average retrieve time for individual container

The number of negotiations (described in 3.2.3) is shown in Figure 13 and Figure 14. Both of the left pictures are data of the first method, and the right ones are for the second method.

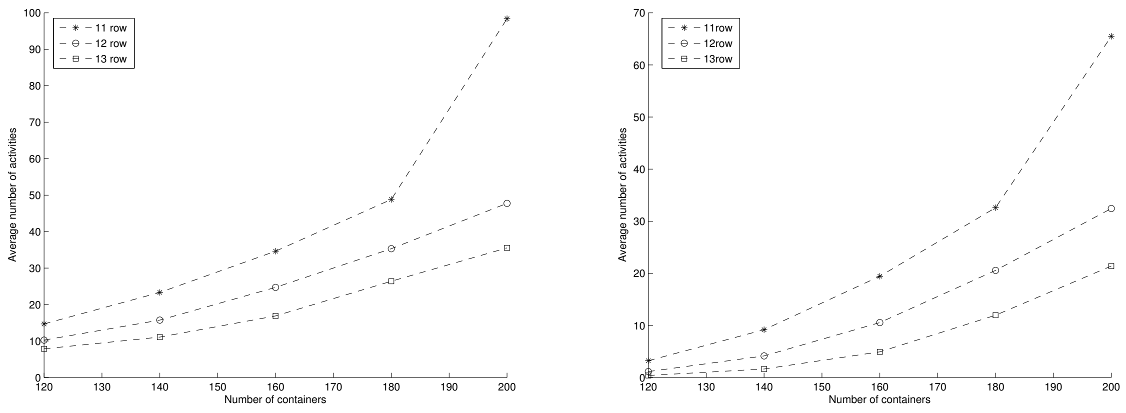

Figure 13: Average number of negotiations initialized by left/right active conveyors

5 Conclusion

A grid-based system is able to receive and distribute π-containers for two trains. Cross-car and cross-train transportation of π-containers can also be completed accurately and timely. The grid-based

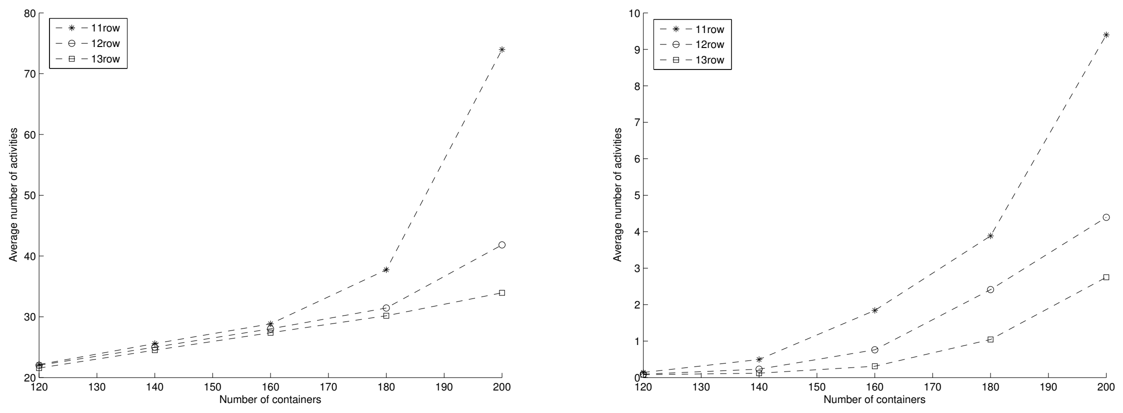

Figure 14: Average number of negotiations initialized by up/down active conveyors

control algorithm also extends the GridStore control algorithm to enable m.ovement in four cardinal directions. As expected, simulation results suggest that performances is highly related to the system's storage density.

References

Eric Ballot and Benoit Montreuil and Collin Thivierge. *Progress in Material Handling Research 2012*. MHIA, Charlote, NC, U.S.A., 2012.

Kevin Gue, Kai Furmans, and Onur Uludag. A High-Density System for Carton Sequencing. In *Proceedings of the 6th International BVL Symposium on Logistis*, Hamburg, Germany, 2012.

Kevin Gue, Kai Furmans, Zazilia Seibold, and Onur Uludag. GridStore: A Puzzle-Based Storage System With Decentralized Control. *IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING*, 11(2), 2014.

Stephan H. Mayer. *Development of a completely decentralized control system for modular continuous conveyors*. PhD thesis, Institutes fr Frdertechnik und Logistiksysteme der Universitt Karlsruhe (TH), Karlsruhe, Baden-Wrttemberg, Germany, April 2009.

Russell D. Meller, Benoit Montreuil, Collin Thivierge, and Zachary Montreuil. Functional Design of Physical Internet Facilities: A Road-Based Transit Center. Technical Report FSA-2013-001, CIRRELT, Montreal, Canada, March 2013.

Benoit Montreuil. Toward a Physical Internet: meeting the global logistics sustainability grand challenge. *Logist. Res.*, 3:71–87, 2011.

Cyrille Pach, Thierry Berger, Emmanuel Adam, Therese Bonte, and Yves Sallez. Proposition of a potential fields approach to solve routing in a rail-road π-hub. In *1st International Physical Internet Conference*, Quebec City, Canada, May 2014.

Yves Sallez, Thierry Berger, Thrse Bonte, and Damien Trentesaux. Proposition of a hybrid control architecture for the routing in a Physical Internet cross-docking hub. In *15th IFAC Symposium on Information Control Problems in Manufacturing*, Ottawa, Canada, June 2015.

Zazilia Seibold, Thomas Stoll, and Kai Furmans. Layout-Optimized Sorting of Goods with Decentralized Controlled Conveying Modules. In *Systems Conference (SysCon), 2013 IEEE International*, pages 628–633. IEEE, April 2013.

Onur Uludag. *GridPick: A High Density Puzzle Based Order Picking System with Decentralized Control*. PhD thesis, Auburn University, Auburn, Alabama, May 2014.