

Optimization of Failure Behavior of a Decentralized High-Density 2D Storage System

Kai Furmans, Kevin R. Gue and Zázilia Seibold

K. Furmans and Z. Seibold
Institute for Material Handling and Logistics, Karlsruhe Institute of Technology,
76131 Karlsruhe, Germany
e-mail: seibold@kit.edu

K. Gue
Department of Industrial and Systems Engineering, Auburn University, Auburn,
AL 36849, USA
e-mail: kevin.gue@auburn.edu

Abstract To meet the requirement of flexibility in intralogistics systems, the GridFlow 2D storage system has been developed which is able react to fluctuating processing volumes and layout changes. It consists of multiple autonomous conveyor modules, FlexConveyors, each equipped with a controller. The decentralized control can be described with a set of rules for the communication by message passing between the controllers of the FlexConveyors. In this paper an adapted control algorithm is presented which enables the conveyor modules to react to occurring failures in neighboring modules. It has been theoretically proved that the presented algorithm prevents system deadlock. Also the impact of occurring failures on the system performance has been examined through a Monte Carlo experiment using discrete event simulation.

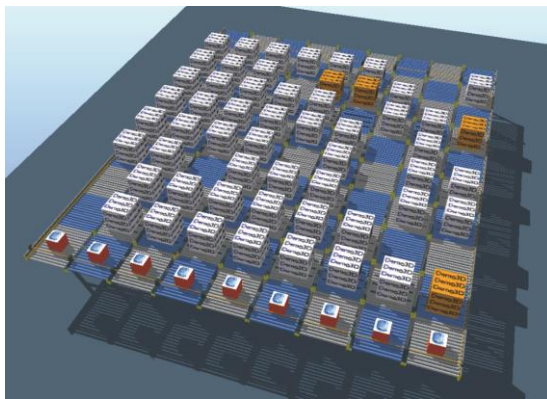


Fig. 1 The GridFlow storage system

1 Introduction

Economic changes like globalization, e-commerce and economic instability – just to name a few – have shifted the conditions and requirements of logistics

systems in recent years. As part of material handling systems, modern storage systems should combine both high performance and flexibility. Decentralized control enables the realization of flexible, automated material handling systems (Furmans et al. 2010, Windt et al. 2007).

To meet the new requirements of high flexibility, the GridFlow system has been developed consisting of multiple, autonomous modules being able to fulfill the tasks of a storage system (Gue et al. 2011) (**Fig. 1**). Each module is equipped with electronics to act independently of a central controller; the control of the GridFlow is decentralized.

GridFlow unifies two different approaches to turn a storage system into a high-performing and space-efficient system. On one hand, the design guarantees high throughput and high density at the same time. On the other hand, it follows the current trend towards flexibility in material flow systems by featuring decentralized control. Single modules can easily be plugged and unplugged in order to adapt to fluctuating volumes and changing layout requirements.

Besides flexibility, another advantage of decentralized control is system reliability: In centralized systems, the failure of the central controller causes a complete system breakdown, whereas, in decentralized systems, the failure of modules does not necessarily affect the function of the other modules.

The research presented in this paper introduces an algorithm enabling conveyor modules in a GridFlow system to react to failures and to resolve blockings in the material flow.

In the following section the general concept of GridFlow is presented. The next section describes the problem of failures in GridFlow and presents the adapted algorithm. We show that the system will not deadlock, and then describe the impact of failures on system performance.

2 System Description

The GridFlow system examined in this paper is designed in 2D with incoming and

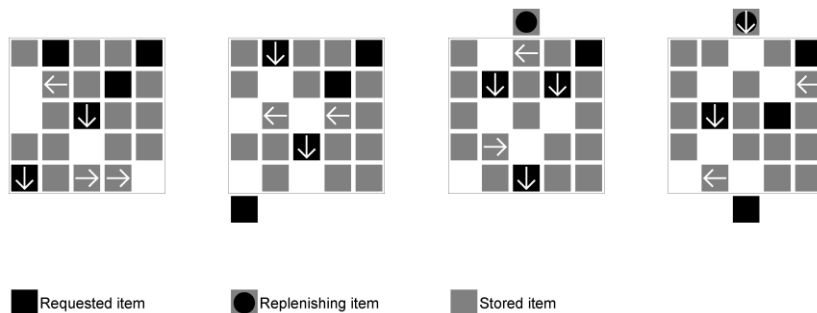


Fig. 2 Example movement in a 5x5 GridFlow system with 4 empty cells (white spaces)

outgoing goods. To enable movement of the goods, there must be at least one empty cell per row in the initial setup. **Fig. 2** shows an example set of movement steps in a 5x5 GridFlow system. As outgoing goods are retrieved to the South of the storage space, requested items move southward to the retrieval row. Also incoming goods, the replenishing items, move southward and fill up the storage space.

To enable southward movement of requested and replenishing items, empty conveyor cells (white spaces) must be south of them. Stored items in the way of requested or replenishing items try to move east- and westward to create empty cells. In **Fig. 2**, white arrows specify the movement of items during four cycles. An item can move to a directly neighboring cell during one cycle.

The decision of which items are moved is made by communication between the controllers in the conveyor units. The conveyors are considered to be agents capable of making decisions based on perceptions and performing resulting actions (Woolridge 2009). Each cycle consists of four process steps, which start in each conveyor unit concurrently.

North-South negotiation:

Communication is conducted in North-South direction. Conveyor units occupied with requested or replenishing items make a decision about conveying. If the South neighbor is empty, the conveyors commit to convey down. If the South neighbor is occupied with a stored item, it will try to transport it east- or westward.

East-West negotiation:

Communication is conducted in East-West direction. The conveyors of each row decide which stored items to transport east- or westward. Occupied conveyors south of requested or replenishing items try to become empty and therefore initiate a message flow following a Request-Willing-Commit logic. Empty cells are needed to enable East-West movement of stored items. Basically, requested and replenishing items located north of the row compete for empty cells.

Transportation:

The conveyors perform the conveying action chosen during the negotiation phases. Active conveyors update their state according to the movement of the conveyed items.

Retrieval, replenishment and requesting:

During this step outgoing goods are retrieved, new items are placed in the replenishment row and new requests are put into the system. Affected conveyors update their state.

To control the system replenishment and the North-South distribution of empty cells, a target row is assigned to each item. The target row defines the storage location to which the item must move. As long as the item has not reached its target row, it is considered to be a replenishing item.

3 Problem Statement

High reliability is one of the advantages of a decentralized system. In order to maximize the reliability of GridFlow, each conveyor must be able to handle failures occurring both to itself and to neighboring conveyors. In this research, only failure types affecting the negotiations between the conveyor units are considered. Therefore, the algorithm has been adapted to the following two failure types: failed conveyors and failed ports.

If a complete conveyor unit fails, the carried item cannot be retrieved. To prevent a system deadlock, the control algorithm has to detect and to resolve blockings of row sections in the storage space. The system is deadlocked if one conveyor having the objective to forward the carried item cannot fulfill its task at any future cycle. The objective of the algorithm is to keep the system working until the failure is repaired and to guarantee that as many requested items as possible can be retrieved.

The following section describes the modification of the algorithm and the countermeasures taken to prevent system deadlock. We then demonstrate that the modified algorithm successfully prevents deadlock, and show the performance of a system with failed conveyors and failed ports. The results can give insights into how reliable a single conveyor unit should work and how quickly failures should be repaired.

4 The Control Concept

A conveyor can either fail mechanically, which means that it can communicate its breakdown to its neighbors, or the controller can fail, which interrupts communication. Both cases have the same consequences on the control problem, only failure detection is done in a different way.

A failed port interrupts message passing and hinders communication between

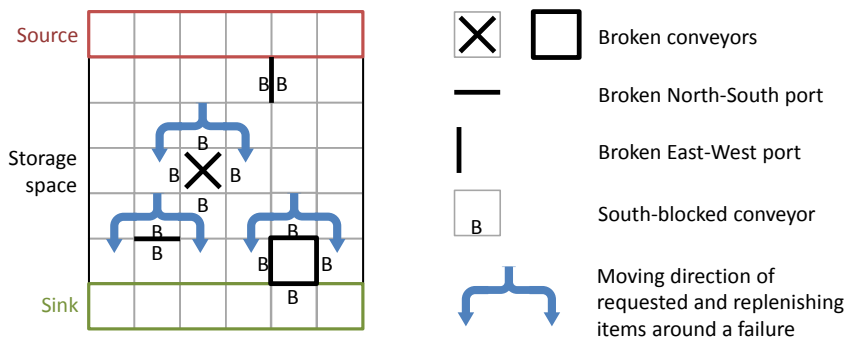


Fig. 3 Schematic representation of a system with failures

the two adjacent conveyor units. The neighbors connected by this port do not get any reply to messages. A failed port is comparable to a passive port at the border of the system because it interdicts conveying in this direction. In the figures below, passive ports are represented by a black line (see **Fig. 3**).

As shown in **Fig. 3** a failed conveyor corresponds to a conveyor whose four ports (North, East, South and West) are unresponsive or passive. The conveyor is blocked on four sides and the currently placed item cannot be moved until the failed conveyor is repaired. In order to generalize, every failure type is represented by a passive port in the control algorithm.

A passive port in the middle of the storage space blocks movement of items: A failed East-West port divides a row into two sections and blocks East-West movement necessary to locate empty cells where they are needed. A failed North-South port hinders South-movement of requested and replenishing items. By implication, requested or replenishing items on conveyors blocked to the South must move east- or westward before being able to move southward.

A conveyor always has one of the states shown in **Table 1**.








Conveyor state	Occupied by	Objective to forward the item
 Empty	-	-
 S-Requesting	Requested or replenishing item	Southward
 Occupied	Stored item	-
 EW-Requesting	Stored item	East- or westward
 S-EW-Requesting	Requested or replenishing item	East-or westward in direction of closest escape to the South

Table 1 Conveyor states

In the following section, message types are introduced which are necessary to detect and resolve the blocking of a row section. The complete algorithm also contains other message types and rules explained in communication protocols depending on negotiation states and message types (Krothapalli et al. 1999).

North-South negotiation

 Request	This message is triggered at the beginning of North-South negotiation.
 Blocked	This message is triggered if a Request cannot be satisfied because of a passive port.

During North-South negotiation, the S-Requesting conveyors send request messages to the South looking for an empty cell. Assuming that the successful reception of a request must be confirmed, the lack of feedback implies a passive port. If

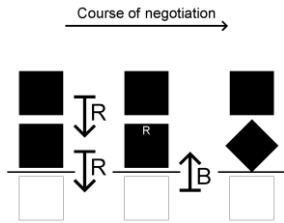


Fig. 4 Example for East-West negotiation

a S-Requesting conveyor detects a passive port, its state turns into S-EW-Requesting. Fig. 4 shows an example.

A S-EW-Requesting conveyor also checks if the closest escape to the South is in the East or the West. If the South-blocked conveyor discovers that there is neither an escape for the carried item in the East nor the West, it does not try to move the item anymore. In this pathological case with a conveyor blocked in three directions, it acts as if it were completely failed.

East-West negotiation

\xrightarrow{R}	Request	This message is triggered at the beginning of East-West negotiation.
\xrightarrow{B}	Blocked	This message is triggered if a conveyor receives a request and movement of the carried box to the opposite side is blocked.
\xrightarrow{RB}	Resolve	This message is triggered if a S-EW-Requesting conveyor is blocked on both blocking sides.

During East-West negotiation, the conveyors decide which items to move east- or

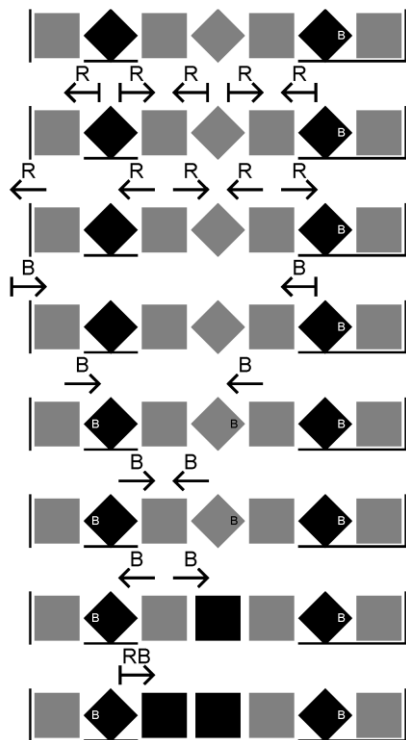


Fig. 5 Example for East-West negotiation

westward. In a system with failures they should additionally detect blocked row sections and take actions to resolve the blocking.

An EW-Requesting conveyor tries to forward the carried item east- or westward to create an empty cell for a requested or replenishing item from the North. To minimize transportation of items, it selects the direction of the closest empty cell enabling East-West movement.

A S-EW-Requesting conveyor also tries to forward the carried item east- or westward. But in contrast to the EW-Requesting conveyor, it should forward the item in direction of the closest escape to the South. This direction has been determined at the end of North-South negotiation. Only if the escape is equidistant in both directions, the conveyor selects the direction of the closest empty cell during East-West negotiation.

At the beginning of East-West negotiation, EW-Requesting and S-EW-

Requesting conveyors initiate Request-messages in one or both directions concurrently. A row section is blocked if the objective of a conveyor to move the carried item east- or westward cannot be satisfied.

To detect passive ports, the successful transmission of a Request-message from conveyor to conveyor must be confirmed. Again, the lack of feedback implies a passive port. A row section is blocked if there is no Empty or S-Requesting conveyor; the requesting conveyor will receive feedback that movement in this direction is not possible. If the row section is blocked in both directions, the conveyor must resolve the blocking by requesting a stored item to move southward.

In the example in **Fig. 5** three conveyors must check if the movement of the carried item is blocked long-term. Two of them actually detect and resolve a blocking: The EW-Requesting conveyor requests the carried item to move southward and the S-EW-Requesting conveyor on the left side requests the stored item on the neighboring conveyor to move southward.

5 Demonstration of System Liveness

Assuming that the retrieval row is empty at the beginning of each cycle, it is only necessary to prove that every requested and replenishing item can be transported to the next row. This is the case if every conveyor with an objective can forward the carried item at some future cycle. Instead of preventing blocked sections, the presented algorithm should guarantee that blockings of row sections are detected and resolved.

Obviously, there are combinations of multiple failures blocking the retrieval of requested items, for example a complete row of failed North-South ports. In this case a conveyor carrying a requested item cannot fulfill its objective and gives it up; it acts as completely failed.

Three questions have to be answered to demonstrate that the described algorithm guarantees that a requested item can be retrieved if there is a path of functioning conveyors and ports to the retrieval row:

- When is a row section blocked?
- How are blocked row sections detected?
- How are blockings of row sections resolved?

Every conveyor has one of the states described in the Table 1.1. An empty or S-Requesting conveyor indicates that a row section is not blocked because it enables East-West movement.

When is a row section blocked?

A row section is blocked if a conveyor with an objective cannot forward the carried item in the required direction. By implication, a row section is blocked if there is an EW-Requesting or S-EW-Requesting conveyor but no empty or S-Requesting conveyor in the conveying direction of the requesting conveyor.

How are blocked row sections detected?

Each conveyor with the objective to move its item east-or westward initiates requests in the potential conveying directions. A request is forwarded by occupied conveyors until it is:

- stopped by an empty or S-Requesting conveyor, because these states indicate that movement is not blocked long-term.
- stopped by an EW-Requesting or S-EW-Requesting conveyor, because this conveyor already sent requests by itself.
- answered with a message that movement is blocked if there is a passive port.

A Blocked-message is forwarded through the complete row until it is stopped either by a conveyor with the indication that the row section is not blocked or by a conveyor that is blocked on the opposite side.

How are blockings of row sections resolved?

A blocking can be resolved by turning a conveyor state from Occupied or EW-Requesting into S-Requesting. The affected item will move southward and leave an empty cell to enable East-West movement. If an EW-Requesting conveyor receives Blocked-messages from both sides, it turns into S-Requesting. If a S-EW-Requesting conveyor is blocked in both directions, it triggers a message to resolve the blocking which is forwarded to the closest conveyor occupied with a stored item.

The message rules defined in communication protocols guarantee the described behavior. Every conveyor with the objective to forward a carried item can fulfill its task at some future cycle. Consequently, every requested item that is not blocked on a failed conveyor can be retrieved, and the system does not deadlock.

6 System Performance Analysis

GridFlow has been modeled in an agent-based, discrete-event simulation environment. Items enter the system by appearing at random cells in the replenishment row and leave the system by disappearing in the retrieval row. In this study the system performance is determined by a CONWIP analysis (CONstant Work In Process), i.e. the number of items and the number of requested items in the system are constant.

For the performance analysis of a system with failures, a high system density has been chosen: The storage space stores 144 items on 12 rows and 13 columns, which corresponds to a density of 92.3 %. The work in process varies from 10 to 130 requests in steps of 10. In order to fulfill the conditions for the CONWIP analysis, the simulation does not exactly correspond to the system behavior in reality: If a requested item is blocked on a failed conveyor, another request is generated in order to keep a constant number of processed requests in the system.

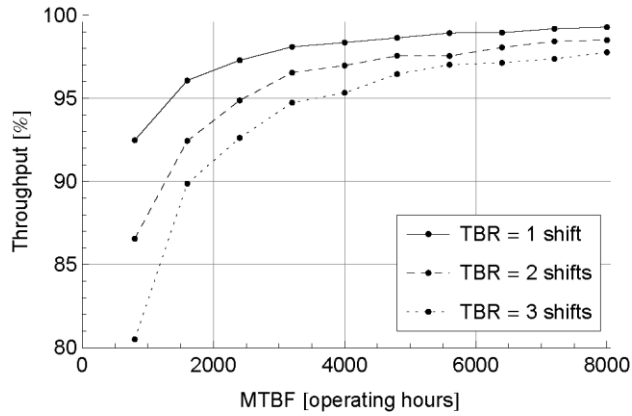


Fig. 6 System performance with statistical failure occurrence

defines the "Mean Time Between Failure" (MTBF) indicating how long a conveyor/port works without failure after being brought into service or after being repaired. In the experiments, MTBF varies from 800 to 8000 operating hours. The probability of a failure occurrence is assumed to be exponentially distributed: A conveyor/port fails with the same probability at all times independently from the last failure occurrence. Basically, every cycle in GridFlow can be understood as Bernoulli trial with $1/\text{MTBF}$ as probability of failure for every conveyor module.

Failed units are not repaired immediately but at a periodically defined time, for example, at the end of a working shift. As the algorithm has been adapted to failures, the system does not need to shut down immediately after failure occurrence but continues to operate with the failure until the next scheduled repair. The "Time Between Repair" (TBR) varies from 1 to 3 shifts. It describes the definite time between two scheduled repairs in contrast to the commonly used term "Mean Time To Repair" (MTTR) describing the statistically distributed time between a failure and its repair.

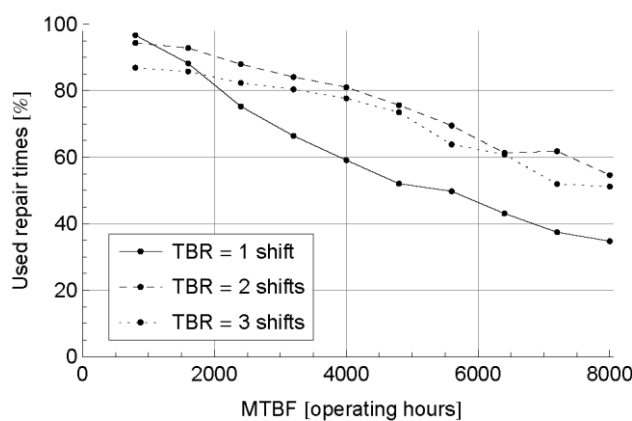


Fig. 7 Usage of repair times

The objective of the study is to simulate the system performance with realistic conditions. The results demonstrate how reliable the conveyor units should be and how frequently failures should be repaired.

Every conveyor unit and every port fails at a certain rate. The reciprocal value of the failure rate defines

A failure can easily be repaired by exchanging the failed conveyor unit with a working one.

Because of the statistical failure occurrence, the interference of multiple failures is highly influencing the result: for example, two failed North-South ports neighboring in

the same row have a significantly greater impact on the throughput than two failed North-South ports in the same column. Therefore, a Monte Carlo simulation with multiple experiments has been conducted for each combination of MTBF, TBR and work in process. The throughput and the number of resolving actions are averaged over the different values of WIP.

Fig. 6 shows the average throughput compared to a system without failures. As expected, the throughput decreases with increasing failure probability ($1/\text{MTBF}$) and decreasing repair frequency ($1/\text{TBR}$). The figure also shows that even with a repair frequency equal to 3 shifts, a good system performance can be achieved despite failures. If, for example, the throughput should not decrease more than 5% compared to a system without failures and if the repair frequency is 3 shifts, a conveyor unit should work at least 4000 operating hours without failure.

Fig. 7 gives insights into the necessity of repairs. It illustrates how often the scheduled repair time is required. For higher MTBF, it is not necessary to schedule a repair after each working shift.

7 Conclusion

The algorithm for the GridFlow system has been successfully adapted to failures in any occurring combination. It has been shown that system deadlock is prevented.

The simulation of a working GridFlow with statistical failure occurrence has shown that good system performance can be realized despite failures. Since it is not required to repair a failure immediately after its detection, high system availability is achieved. Because a failure can be repaired by exchanging the failed unit, a relatively high repair frequency has been assumed. Realizing a high technical reliability would decrease the required repair frequency.

References

- Furmans K, Schönung F, Gue K R (2010) Plug-and-work material handling systems http://web.mac.com/krgue/Kevin_Gue/Blog/Entries/2010/8/25_Plug-and-Work_Material_Handling_files/futuremh.pdf. Accessed 11 Nov 2011
- Gue K R, Furmans K (2011) Decentralized Control in a Grid-Based Storage System. Proceedings of the 2011 Industrial Engineering Research Conference, eds. T. Doolen and E. Van Aken, 2011
- Krothapalli N K C, Deshmukh A V (1999) Design of negotiation protocols for multi-agent manufacturing systems. International Journal of Production Research 37 (7), 1601-1624
- Windt K, Hülsmann M (2007) Changing paradigms in logistics - Understanding the shift from conventional control to autonomous cooperation and control. Springer Berlin Heidelberg
- Woolridge M (2009) An Introduction to MultiAgent Systems. John Wiley & Sons Ltd