Received: 15 January 2024 | Revised: 5 April 2024 | Accepted: 6 May 2024 | Published online: 13 May 2024

RESEARCH ARTICLE

Journal of Data Science and Intelligent Systems
2024, Vol. 2(4) 241–254
DOI: 10.47852/bonviewJDSIS42022471

BON VIEW PUBLISHING

Efficient Scheduling of Data Transfers in Multi-tiered Storage

Nan Noon Noon[1,*], **Janusz Getta**[1] **and Tianbing Xia**[1]

[1]*School of Computing and Information Technology, University of Wollongong, Australia*

Abstract: Multi-tiered persistent storage systems integrate many types of persistent storage devices, such as different types of NVMes, SSDs, and HDDs. This integration provides a multi-level view of persistent storage, where each tier has a different data transmission speed and capacity. Data transfer processes operating on multi-tiered persistent storage allow for the parallelisation of data transfers among the partitions of data at the same or different tiers. This work considers the problem of efficient scheduling of parallel data transfers between the tiers of persistent storage. We consider a data processing model where several data transfer processes move or copy data from one tier to another through the buffers in transient memory. We propose a new model for data processing over multi-tiered persistent storage and new algorithms to minimise both the overall time spent on parallel data transfers and the idle time of data transfer processes. We also describe how the scheduling algorithms dynamically apply different procedures to assign data transfers to the processes. Finally, we present the outcomes from the experiments that confirm the correctness and efficiency of the scheduling algorithms.

Keywords: multi-tiered persistent storage, parallel data processing, data transfers, scheduling, resource management, parallel data distribution, data transfer processes

1. Introduction

The fast-paced growth of data analysis techniques in recent years has accumulated a vast amount of data. As a result, organizations are looking for higher capacity and faster persistent storage solutions. They use various storage devices to manage costs and outsource storage and data management to cloud service providers offering diverse storage options. However, the complexity of integrating different storage devices and optimizing data processing presents a challenge.

A multi-tiered view of persistent storage unifies many types and characteristics of persistent storage devices into a single logical device [1]. A multi-tiered approach to persistent storage is used to simplify data processing. This approach consolidates various storage devices into a hierarchical structure based on capacity and access time characteristics. Data can be transferred between tiers, optimizing access times and storage utilization. However, resource allocation becomes crucial due to the limited capacity of higher tiers and the need to consider factors like processing speed and access frequencies.

One solution to expedite data processing is parallelizing data transfer across storage tiers. This paper focuses on solving the issue of scheduling parallel data transfers within multi-tiered persistent storage systems. We propose a model where data transfers are executed by multiple processors, aiming to minimize transfer time and processor idle time.

The paper's key contributions are as follows:

1) Introducing a novel model for query processing over multi-tiered persistent storage, organizing queries into sequences of data transfer sets.
2) Presenting new scheduling algorithms to optimize parallel data transfers, reducing overall processing time.
3) Discussing dynamic scheduling strategies that adapt to changes in workload characteristics.
4) Showing how well the suggested algorithms work in creating efficient plans for transferring data in parallel.

By addressing these challenges, this research enhances data processing efficiency in multi-tiered persistent storage systems, paving the way for improved resource utilization and performance.

The paper is organized as follows: Section 2 presents earlier works related to the organization of multi-tiered storage and the scheduling of data transfers, and Section 3 includes notational conventions and the definitions of new concepts. Section 4 then explains the scheduling algorithms, and Section 5 presents the experiments and interpretations of the results. Finally, Section 6 summarizes the paper, lists the conclusions, and explains the avenues for future research in the same area.

2. Previous Works

Integrating additional layers of devices and memory can significantly enhance computers' computing and I/O performance [1]. In today's rapidly evolving technological landscape, I/O performance is increasingly significant across various applications. Artificial intelligence techniques like machine learning and deep learning rely on processing diverse data types and structures,

*__Corresponding author:__ Nan Noon Noon, School of Computing and Information Technology, University of Wollongong, Australia. Email: nnn326@uowmail.edu.au necessitating efficient data storage and access across multiple locations [2].

Organizations increasingly adopt multi-cloud systems to ensure seamless data access anytime and anywhere [3]. However, with the escalating data growth, more than relying on cloud storage may be required, leading to adopting hybrid multi-cloud solutions [3]. Concurrently, the market introduces multi-tiered storage systems [4, 5], offering varied read/write speeds, features, and prices, including HDDs, SSDs, and NVMe. NVMe stands out for its rapid data processing capabilities [6–8], prompting widespread adoption of multi-tiered persistent storage models among organizations [9, 10].

This research leverages multi-tiered persistent storage with partitions to efficiently manage data and facilitate parallel processing [11]. However, the adoption of faster devices may be limited by cost and storage capacity constraints, necessitating optimal resource allocation and data transfer planning over multi-tiered storage systems [12].

Our work builds upon previous studies addressing data distribution and resource allocation in multi-tiered persistent storage systems [13], aiming to improve database management system performance. Various scheduling theories and methods have been proposed to address resource allocation challenges, ranging from dynamic programming algorithms to quality-aware scheduling techniques [14–17].

These scheduling methods, including EDF, SJF, and PAQRS, offer different approaches to optimizing service quality and data quality [18, 19]. Each method presents its unique advantages and limitations, necessitating careful consideration of workload characteristics and system requirements. Thus, our study aims to contribute to this evolving field by addressing the challenges of resource allocation and scheduling in multi-tiered persistent storage systems.

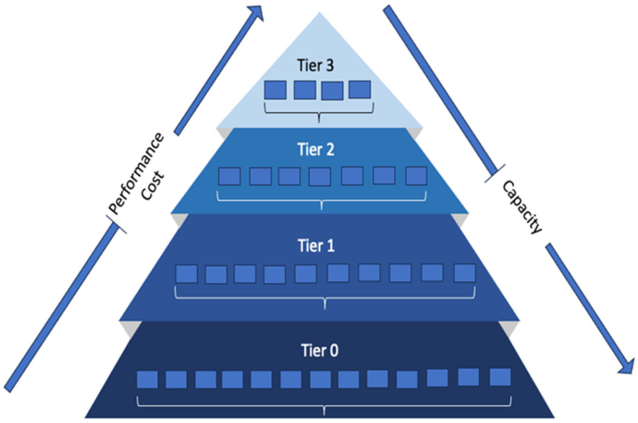
3. Data Processing Model

3.1. Architecture of multi-tiered persistent storage

The sequence of persistent storage tiers (levels) in a multi-tiered persistent storage system is denoted by $\boldsymbol{L} = <l_0, \ \ldots, \ l_n>$. The slowest tier with the lowest performance and the highest capacity is represented by l_0, while the fastest tier with the highest performance and the lowest capacity is represented by l_n. Each tier l_i, where i is between 1 and n, is implemented as a set of persistent storage devices $D_i = \{d_{i1}, \ \ldots, \ d_{im}\}$. Any device located at the same tier l_i is described by a triple (r_i, w_i, s_i), where r_i is the reading speed of the tier, w_i is the writing speed at the tier, and s_i is the total amount of storage available on the device. Further on, we refer to a persistent storage device in D_i as a *partition* at a tier l_i.

In a model of multi-tiered persistent storage extended with partitions, data can be read or written at the same tier and at the same time from/to different partitions/devices. A simplified structure of multi-tiered persistent storage considered in this paper is presented in Figure 1.

Figure 1
Architecture of multi-tiered persistent storage



3.2. Extended Petri Net

In this work, we consider a data processing model where a set of queries, $\{q_1, \ \ldots, \ q_m\}$, is submitted to a database system for simultaneous processing. For each query in $\{q_1, \ \ldots, \ q_m\}$, a cost-based query optimizer generates several query processing plans and, for each query, selects the plan with the lowest processing costs. Next, each query processing plan is converted into an *Extended Petri Net* (*EPN*) to represent the operations on data and data flows that can be processed simultaneously. The *EPN* is a quadruple $(B, \ V, \ E, \ W)$, where $B = \{b_1, \ \ldots, \ b_j\}$ is a set of input/output datasets, $V = \{v_1, \ \ldots, \ v_j\}$ is a set of operations, $E = \{e_1, \ \ldots, \ e_j\}$ is a set of edges representing dataflows, and W is a function $W\!: E \rightarrow N^+$ that assigns numerical values to each edge in E.

A set $b \in B$ is a set of pairs $\{(D_i, \ l_j.d_k), \ \ldots, \ (D_j, \ l_k.d_l)\}$, where each D_i is the total number of data blocks available at a partition $l_j.d_k$. Each edge $e \in E$ is either a pair $(b, \ v)$ or a pair $(v, \ b)$ where $b \in B$ and $v \in V$. A sample *EPN* is given in Figure 3.

Example 1 The present example considers the following query.

```
SELECT CCUSTKEY, CNAME, OORDERDATE
FROM (
SELECT *
FROM CUSTOMER
WHERE CNATIONKEY BETWEEN 1 AND 6
UNION
SELECT *
FROM CUSTOMER
WHERE CNATIONKEY BETWEEN 10 AND 15
) as T1
JOIN ORDER
ON CCUSTKEY = OCUSTKEY
WHERE OORDERDATE < 2000-01-01;
```

A sample query processing plan obtained from the application of EXPLAIN PLAN statement to a query above is visualized in Figure 2. The Extended Petri Net $(B, \ V, \ E, \ W)$ for the query processing plan is given in Figure 3.

3.3. Data transfer

An *EPN* is converted into a sequence of sets of data transfers $Q = <\{(q, \ l_j.d_n \ \tau_1), \ \ldots, \ (q, \ l_k.d_j, \ \tau_i)\}, \ \ldots, \ \{(q, \ l_x.d_y, \ \tau_j), \ \ldots, \ (q, \ l_y.d_i, \ \tau_k)\}>$. The detailed procedure for converting a query into an *EPN* is presented in Noon et al. [13]. A triple $(q, \ l_j.d_n, \ \tau_k)$ denotes a single data transfer. The first element, q, represents a query to which the transfer belongs; the second element $l_j.d_n$ represents a tier l_j and partition/device d_n involved in the transfer; and the last element, τ_k, represents the total number of time units required to process the transfer.

Information from a query processing plan regarding the total amount of data involved in each operation and the speed of read and write operations at each persistent storage tier is used to predict the processing time τ_k of a transfer.

The total number of data transfers depends on the size of a buffer. For example, reading 100 data blocks from device d_j at tier

Figure 2
Visualization of a query processing plan

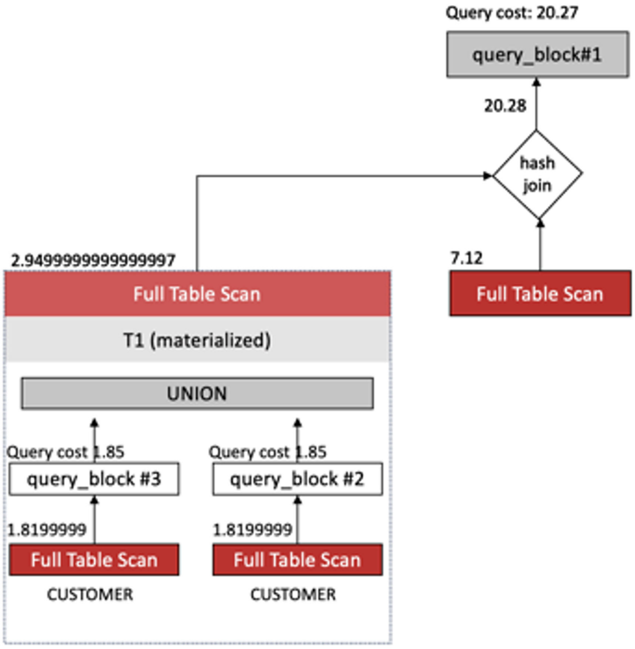
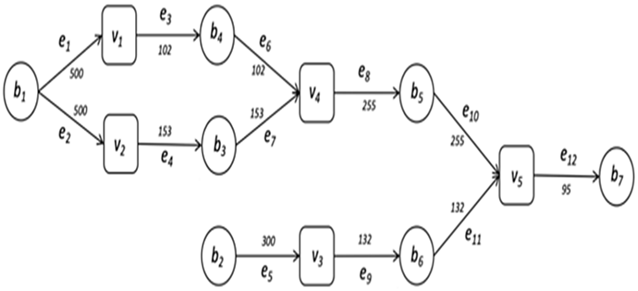


Figure 3
Extended Petri Net



l_j and writing them to device d_i at tier l_j with a buffer size of 50 data blocks requires two transfers to read data and two transfers to write data.

3.4. Data transfer processes

Several dedicated data transfer processes copy data across the partitions in multi-tiered persistent storage and transient data buffers. A data transfer process either reads data from the persistent storage or writes it to a data buffer or vice versa. We assume that there are m processes $\boldsymbol{P} = \{P_1, \dots, P_m\}$ available for the implementation of data transfers. Data transfer processes operate simultaneously on the tiers of persistent storage and transient memory. For each $1 \leq i \leq m$, a *data transfer plan* π_i is assigned to a data transfer process, and P_i is a sequence of data transfers $<(q_i, l_j.d_k\ \tau_i), \dots, (q_j, l_x.d_y, \tau_k)>$.

3.5. Candidate data transfers

A *candidate data transfer* in an implementation of a query q is a data transfer in Q that has not yet been assigned to any process and therefore belongs to the first set of data transfers in Q that includes at least one data transfer not assigned to any data transfer process.

Example 2 In this example, we consider the queries q_1, q_2, and q_3 implemented into the following sequences of sets of data transfers:

- Q_1: $<\{(q_1, l_4.d_1, 8)\}, \{(q_1, l_3.d_3, 9)\}>$
- Q_2: $<\{(q_2, l_3.d_2, 3), (q_2, l_2.d_1, 4)\}, \{(q_2, l_1.d_1, 3)\}, \{(q_2, l_3.d_1, 7)\}>$
- Q_3: $<\{(q_3, l_1.d_2, 3), (q_3, l_4.d_1, 3)\}, \{(q_3, l_1.d_1, 7), (q_3, l_3.d_1, 3)\}, \{(q_3, l_1.d_2, 3)\}, \{(q_3, l_1.d_1, 5)\}>$

Notably, at the very beginning, none of the data transfers implementing the queries is assigned to a data transfer process. As a result, we can combine all the first sets of transfers from each query and form a set of candidate transfers. A set of candidate transfers T_c for Q_1, Q_2, and Q_3 includes the following sets of transfers: $\{\{(q_1, l_4.d_1, 8)\}, \{(q_2, l_3.d_2, 3), (q_2, l_2.d_1, 4)\}, \{(q_3, l_1.d_2, 3), (q_3, l_4.d_1, 3)\}\}$.

4. Scheduling Algorithms

4.1. Overview

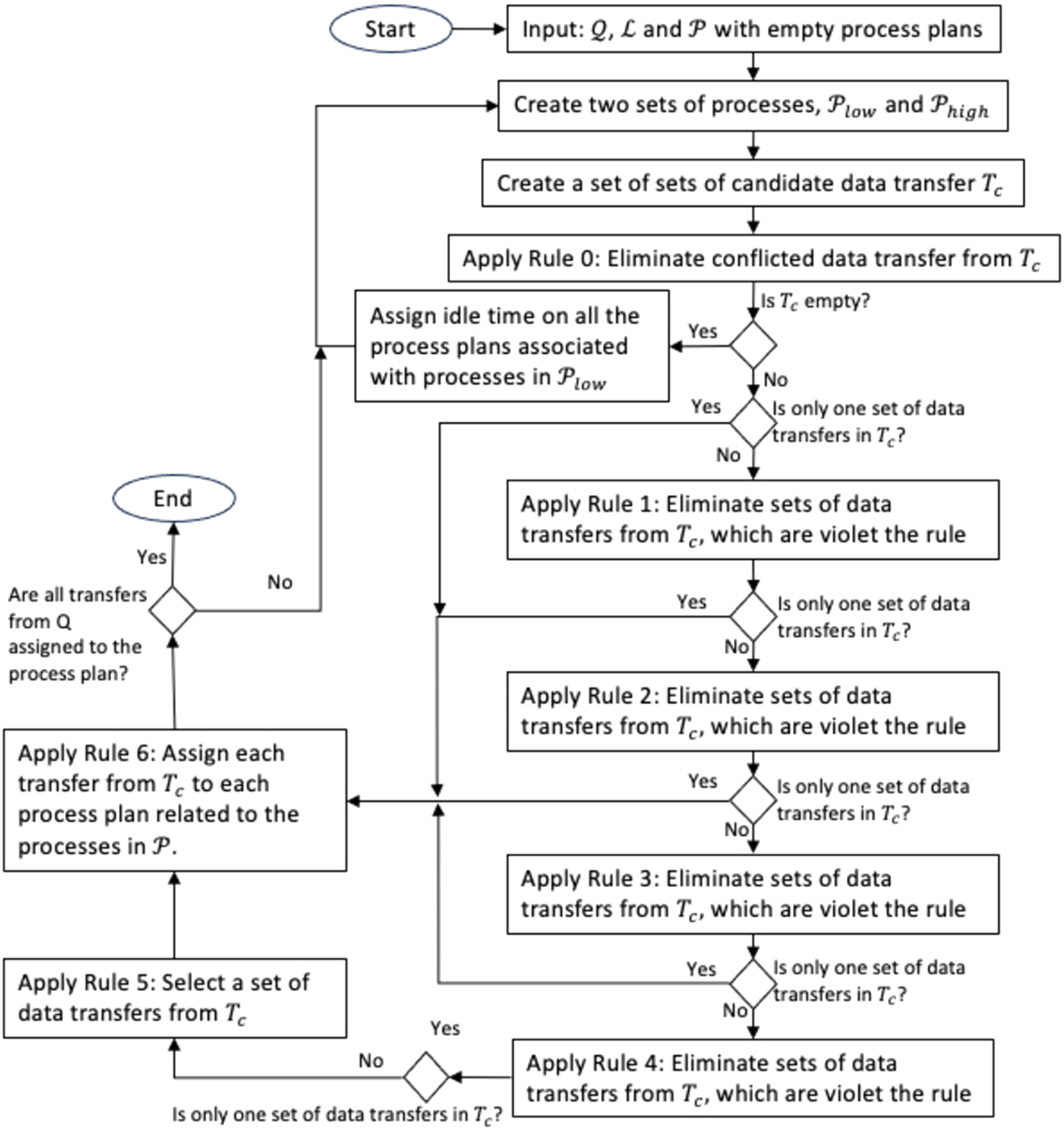
Algorithm 1 is the central point linking all other algorithms that assign the data transfers to the transfer processes and allocate resources over multi-tiered persistent storage. A flowchart of *Algorithm 1* is given in Figure 4. The inputs to *Algorithm 1* include a set of queries converted into the sequences of sets of data transfers $\boldsymbol{Q} = \{Q_1, \dots, Q_n\}$, a set of processes $\boldsymbol{P} = \{P_1, \dots, P_m\}$, a sequence of persistent storage tiers $\boldsymbol{L} = < l_0, \dots, l_n >$, and a threshold value delta that identifies very long data transfers. An output from *Algorithm 1* is a set of data transfer plans $\{\pi_1, \dots, \pi_{\text{m}}\}$ assigned to each data transfer process.

Algorithm 1 iteratively processes the sequences of sets of data transfers in $\boldsymbol{Q}$. First, the algorithm applies *Algorithm 2* to identify a subset of processes in $\boldsymbol{P}$, denoted as $\boldsymbol{P_{low}}$, with the lowest workload presently assigned, and a subset of the remaining processes, with larger workloads, denoted as $\boldsymbol{P_{high}}$. The procedure that finds a set of processes with a lower workload is described in Section 4.2. Following this, the algorithm generates a set of candidate transfer sets T_c using *Algorithm 3* from the transfers in $\boldsymbol{Q}$ that are yet to be assigned to a process. The procedure that creates T_c is described in Section 4.3.

Next, *Algorithm 1* applies the scheduling rules to narrow down T_c to either a single set of data transfers or an empty transfer. An empty transfer represents an idle period when none of the candidates in T_c can be assigned to the selected data transfer process. When more than one set of data transfers is found in T_c, the algorithm applies the scheduling rules to narrow down candidate transfers from T_c.

Algorithm 4 applies *Rule 0* to remove conflicting data transfers from T_c. A conflict between two data transfers occurs when both transfers attempt to access the same partition at the same moment

Figure 4
A flow chart of Algorithm 1


Start
Input: $\mathcal{Q}$, $\mathcal{L}$ and $\mathcal{P}$ with empty process plans
Create two sets of processes, $\mathcal{P}_{low}$ and $\mathcal{P}_{high}$
Create a set of sets of candidate data transfer T_c
Apply Rule 0: Eliminate conflicted data transfer from T_c
Is T_c empty?
Yes
Assign idle time on all the process plans associated with processes in $\mathcal{P}_{low}$
No
Yes
Is only one set of data transfers in T_c?
No
Apply Rule 1: Eliminate sets of data transfers from T_c, which are violet the rule
End
Yes
Are all transfers from Q assigned to the process plan?
No
Yes
Is only one set of data transfers in T_c?
No
Apply Rule 2: Eliminate sets of data transfers from T_c, which are violet the rule
Apply Rule 6: Assign each transfer from T_c to each process plan related to the processes in $\mathcal{P}$.
Yes
Is only one set of data transfers in T_c?
No
Apply Rule 3: Eliminate sets of data transfers from T_c, which are violet the rule
Apply Rule 5: Select a set of data transfers from T_c
Yes
Is only one set of data transfers in T_c?
No
Yes
No
Apply Rule 4: Eliminate sets of data transfers from T_c, which are violet the rule
Is only one set of data transfers in T_c?

in time or when their processing order violates an order indicated by a query plan. If more than one set of data transfers is found in T_c, then Algorithms 5–10 apply the following procedures:

- *Algorithm 5* applies *Rule 1* to retain only significantly long data transfers. This rule minimizes long idle times for future allocation.
- *Algorithm 6* applies *Rule 2* to retain only the data transfers associated with the sequences with the longest processing time. This rule minimizes overall processing time and balances the assigned workload among many data transfer processes.
- *Algorithm 7* applies *Rule 3* to retain only data transfers with a higher chance of conflict in the near future. This rule is intent to minimize conflict in future allocations.
- *Algorithm 8* applies *Rule 4* to retain only the data transfers associated with the sequences with the largest number of data transfers.
- *Algorithm 9* applies *Rule 5* to randomly select a set of transfers from the remaining transfers in T_c.
- *Algorithm 10* assigns the sets of data transfers in T_c to the processing plans for each process in $\boldsymbol{P}$ using *Rule 6*.

At the end of the iteration, *Algorithm 1* removes all data transfers from $\boldsymbol{Q}$ that are included in T_c and updates a set of processes $\boldsymbol{P}_{low}$ and $\boldsymbol{P}_{high}$ using *Algorithm 2*.

Thereafter, the iteration described above is repeated until all data transfers in $\boldsymbol{Q}$ are assigned to the transfer processes. Finally, the algorithm returns a set of data transfer plans $\{\pi_1, \ldots, \pi_m\}$ assigned to each data transfer process.

Algorithm 1: Main Algorithm

```
Require: A set of processes $\boldsymbol{P}$, a set of sequences $\boldsymbol{Q}$, a sequence of tiers $\boldsymbol{L}$, a set of devices associated with each tier $\boldsymbol{D}$, a threshold value δ to determine the significantly long sequences, and the number of sets is $f$ to analyze a future conflict within the selected $f$ sets from each sequence.
Ensure: A set of data transfer process plans $\{\pi_1, \ldots, \pi_m\}$.
1: Let the total number of processes be $p$ and create $p$ number of empty data transfer plans $\pi_1, \ldots, \pi_p$; associate each plan with a process in $\boldsymbol{P}$.
2: Pass the information of a set of processes $\boldsymbol{P}$ and a set of plans $\{\pi_1, \ldots, \pi_m\}$ to Algorithm 3 to obtain a set of lower workload processes $\boldsymbol{P_{low}}$ and $\boldsymbol{P_{high}}$.
3: Pass a set of $\boldsymbol{Q}$ and a set of data transfer plans $\{\pi_1, \ldots, \pi_p\}$ to Algorithm 3 to obtain a set of sets of candidate data transfers $T_c$.
4: To eliminate conflict from $T_c$, the algorithm passes the information of $\boldsymbol{P_{low}}$, $\boldsymbol{P_{high}}$, and $T_c$ to Algorithm 4.
5: if $T_c$ is empty then
6:     Add idle time unit to all process plans associated with processes in $\boldsymbol{P_{low}}$.
7:     Go back to line 2.
8: end if
9: if $T_c$ has more than one set of data transfers then
10:    Apply Rule 1 by using Algorithm 5 with $T_c$ and threshold value $\delta$.
11:    Algorithm 5 eliminates data transfers from $T_c$ that fail to comply with Rule 1.
12:    if $T_c$ has more than one set of data transfers then
13:        Apply Rule 2 by using Algorithm 6 with $T_c$, a set of sequences $\boldsymbol{Q}$.
14:        Algorithm 6 eliminates data transfers from $T_c$ that fail to comply with Rule 2.
15:        if $T_c$ has more than one set of data transfers then
16:            Apply Rule 3 by using Algorithm 7 with $T_c$, a set of sequences $\boldsymbol{Q}$.
17:            Algorithm 7 eliminates data transfers from $T_c$ that fail to comply with Rule 3.
18:            if $T_c$ has more than one set of data transfers then
19:                Apply Rule 4 by using Algorithm 8 with $T_c$, a set of sequences $\boldsymbol{Q}$.
20:                Algorithm 8 eliminates data transfers from $T_c$ that fail to comply with Rule 4.
21:                if $T_c$ has more than one set of data transfers then
22:                    Apply Rule 5 by using Algorithm 9 with $T_c$.
23:                    Algorithm 9 eliminates data transfers from $T_c$ that fail to comply with Rule 5.
24:                end if
25:            end if
26:        end if
27:    end if
28: end if
29: Assign each data transfer from $T_c$ to each process plan by using Rule 6 in Algorithm 10 that returns an updated set of process plans $\{\pi_1, \ldots, \pi_p\}$.
30: Remove all data transfers from $\boldsymbol{Q}$ that are included in $T_c$.
31: if $\boldsymbol{Q}$ is not empty then
32:    Go back to line 2.
33: else
34:    Return an updated set of data transfer plans $\{\pi_1, \ldots, \pi_p\}$.
35: end if
```

4.2. Finding the processes with the lowest workload assigned

This section explains how to divide a set of processes $\boldsymbol{P}$ into two groups. The first group, called $\boldsymbol{P_{low}}$ consists of processes with the lowest workload. The remaining processes are assigned to $\boldsymbol{P_{high}}$. To find a process with the lowest workload assigned, we calculate the length of the data transfer plan assigned to each process. The length of a plan π is denoted as $\boldsymbol{T}(\pi)$ where $\pi = <(q_i, l_j.d_k, \tau_i), \ldots, (q_j, l_x.d_y, \tau_k)>$ is computed with Equation (1) below:

$$T(p) = t_i + \ldots + t_k \tag{1}$$

As input, *Algorithm 2* takes a set of data transfer processes $\boldsymbol{P}$ and the current state of data transfer plans $\{\pi_1, \ldots, \pi_j\}$ associated with the processes. The algorithm computes the length of workload for each plan and then saves the processes with the lowest assigned workload in $\boldsymbol{P_{low}}$. The remaining processes are saved in $\boldsymbol{P_{high}}$. *Algorithm 2* returns the two sets of data transfer processes $\boldsymbol{P_{low}}$ and $\boldsymbol{P_{high}}$.

Algorithm 2: Finding the processes with high and low workloads assigned

```
Require: A set of processes $\boldsymbol{P} = \{P_1, \ldots, P_j\}$ and a set of plans $\{\pi_1, \ldots, \pi_j\}$.
Ensure: A set processes $\boldsymbol{P_{low}}$ with the lowest workload assigned and a set of remaining processes $\boldsymbol{P_{high}}$.
for each plan in $\{\pi_1, \ldots, \pi_j\}$ do
    Let the current plan be $\pi_i = < (q_j, l_i.d_j, \tau_x), \ldots, (q_k, l_j.d_k, \tau_y)>$.
    Use Equation (1) to compute the length for $\pi_i$, $\boldsymbol{T}(\pi_i)$.
end for
Find the smallest value $\boldsymbol{T}_{low}$ from $\boldsymbol{T}(\pi_1), \ldots, \boldsymbol{T}(\pi_j)$.
Save all processes with the lowest workload $\boldsymbol{T}_{low}$ in $\boldsymbol{P_{low}}$.
Save all remaining processes in $\boldsymbol{P_{high}}$.
Return $\boldsymbol{P_{low}}$ and $\boldsymbol{P_{high}}$.
```

An example below shows how to split a set of processes $\boldsymbol{P}$ into $\boldsymbol{P_{low}}$ and $\boldsymbol{P_{high}}$.

Example 3 In this example, we consider the data transfer processes $\boldsymbol{P} = \{P_1, P_2, P_3, P_4, P_5\}$ and the data transfer plans $\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5\}$ assigned to the respective processes.

- $\pi_1 = <(q_1, l_1.d_3, 10)>$
- $\pi_2 = <(q_3, l_2.d_2, 3)>$
- $\pi_3 = <(q_2, l_1.d_1, 6)>$
- $\pi_4 = <(q_2, l_2.d_1, 5), (q_1, l_1.d_2, 3)>$
- $\pi_5 = <(q_3, l_2.d_3, 3)>$

The workload for P_1 is $\boldsymbol{T}(\pi_1) = 10$, for P_2 is $\boldsymbol{T}(\pi_2) = 3$, for P_3 is $\boldsymbol{T}(\pi_3) = 6$, for P_4 is $\boldsymbol{T}(\pi_4) = 8$, and for P_5 is $\boldsymbol{T}(\pi_5) = 3$.

A set of the lowest workload processes as $\boldsymbol{P_{low}} = \{P_2, P_5\}$.

The remaining processes are included in $\boldsymbol{P_{high}} = \{P_1, P_3, P_4\}$.

4.3. Finding candidate transfers

Algorithm 3 uses a set of sequences $\boldsymbol{Q} = \{Q_1, \ldots, Q_n\}$ to find candidate data transfers. The algorithm iterates over $\{Q_1, \ldots, Q_n\}$ and inserts the first set from each Q_i, $i = 1, \ldots, n$ into T_c.

Algorithm 3: Finding candidate transfers T_c

```
Require: A set of sequences $\boldsymbol{Q} = \{Q_1, \ldots, Q_n\}$.
Ensure: A set of sets of candidate data transfers $T_c = \{\{(q_i, l_j.d_k, \tau_k), \ldots, (q_j, l_k.d_j, \tau_i)\}, \ldots, \{(q_k, l_i.d_n, \tau_j), \ldots, (q_l, l_n.d_i, \tau_n)\}\}$.
1: for $Q_i \in \{Q_1, \ldots, Q_n\}$ do
2:     Get the first set of data transfers from $Q_i$ and append it to $T_c$.
3: end for
4: Finally, return the result $T_c = \{\{(q_i, l_j.d_k\ \tau_k), \ldots, (q_j, l_k.d_j\ \tau_i), \ldots, (q_k, l_i.d_n\ \tau_j), \ldots, (q_l, l_n.d_i\ \tau_n)\}\}$.
```

4.4. Scheduling rules

4.4.1. Eliminating conflicting data transfers

Algorithm 4 applies *Rule 0* to eliminate the conflicts between data transfers already assigned to the processes and the candidate data transfers in T_c. *Algorithm 4* operates on the sets of data transfer processes $\boldsymbol{P}_{low}$ and $\boldsymbol{P}_{high}$ along with their assigned data transfer plans and on a set of sets of candidate data transfers T_c. The algorithm finds the size of lower workload $\boldsymbol{T}_{low}$ assigned to one of the processes in $\boldsymbol{P}_{low}$ and the size of the highest workload $\boldsymbol{T}_{high}$ assigned to one of the processes in π. Next, the algorithm computes the time range F as in the following way.

$$F = [(\boldsymbol{T}_{low} + 1), \boldsymbol{T}_{high}] \qquad (2)$$

The algorithm then retrieves the transfers that are at the tail of each plan associated with processes from $\boldsymbol{P}_{high}$ and that overlap with a timeframe F. All such transfers are added to a set of data transfers T_a.

In the next step, the algorithm identifies the conflicts between the candidate transfers in T_c and the transfers in T_a. The data transfers $(q_i, l_m.d_j, \tau_i) \in T_a$ and $(q_j, l_n.d_k, \tau_j) \in T_c$ conflict when one of the following conditions is satisfied:

- **Condition 1:** The transfers belong to the same query, $q_i = q_j$, and their processing order differs from the order determined by a sequence of transfers implementing a query q_i.
- **Condition 2:** The transfers belong to different queries $q_i \neq q_j$, and both transfers are related to the same device, such as $l_m.d_j = l_n.d_k$.

Algorithm 4: Eliminating conflicting data transfers (*Rule 0*)

```
Require: A set of processes $\boldsymbol{P}_{low}$ with the data transfer plans assigned, a set of processes $\boldsymbol{P}_{high}$ with the data transfer plans assigned, and a set of candidate data transfer sets $T_c$.
Ensure: A set of sets of candidate transfers $T_c$ with no conflicts.
1: Create an empty set of data transfers $T_a = \{\ \}$.
2: Select a process at random from the set $\boldsymbol{P}_{low}$, and retrieve the corresponding plan, denoted as $\pi_i$.
3: Calculate the amount of time needed to carry out the plan $\pi_i$; use Equation (1) and refer to it as $\boldsymbol{T}(\pi_i)$.
4: Select a highest workload process from $\boldsymbol{P}_{high}$, and retrieve the corresponding plan, denoted as $\pi_j$.
5: Calculate the amount of time needed to carry out the plan $\pi_j$; use Equation (1) and refer to it as $\boldsymbol{T}(\pi_j)$.
6: Next, compute $F$ by using Equation (2).
7: for each process in $\boldsymbol{P}_{high}$ do
8:     Let the current process be $P_j$ and associated plan be $\pi_j$.
9:
```

(*Continued*)

(*Continued*)

```
9:     Select all the data transfers from a tail of $\pi_j$ that fall within the time range of $F$ and append them to $T_a$.
10: end for
11: for each set in $T_c$ do
12:     Let the current set be $\{(q_i, l_j.d_k, \tau_y), \ldots, (q_i, l_n.d_j, \tau_k)\}$.
13:     for each data transfer in the current set do
14:         Let the current data transfer be $(q_i, l_j.d_k, \tau_y)$.
15:         for each data transfer in $T_a$ do
16:             Let the current data transfer be $(q_j, l_n.d_k, \tau_j)$.
17:             if conflict is found between $(q_i, l_j.d_k, \tau_y)$ and $(q_j, l_n.d_k, \tau_j)$ by Condition 1 then
18:                 Remove the current set from $T_c$.
19:                 Exit the loop and return to line 11.
20:             else
21:                 if Conflict is found between $(q_i, l_j.d_k, \tau_y)$ and $(q_j, l_n.d_k, \tau_j)$ by Condition 2 then
22:                     Remove $(q_i, l_j.d_k, \tau_y)$ from $T_c$.
23:                 end if
24:             end if
25:         end for
26:     end for
27: end for
28: Finally, return a conflict-free set of sets of candidate data transfers $T_c$.
```

Example 4 The following is a sample trace from the processing of *Algorithm 4*. Consider three data transfer processes P_1, P_2, *and* P_3 and three queries $\boldsymbol{Q} = \{Q_1, Q_2, Q_3\}$.

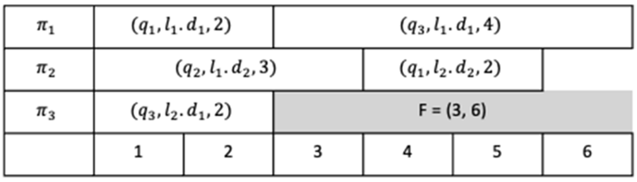
Assume that the original sequences are as follows:
Q_1: $<\{(q_1, l_1.d_1, 2), (q_1, l_2.d_2, 2)\}\ \{(q_1, l_3.d_1, 3)\}, \{(q_1, l_2.d_1, 3)\}>$
$Q2$: $<\{(q_2, l_1.d_2, 3), (q_2, l_2.d_1, 4)\}, \{(q_2, l_3.d_1, 3)\}>$
$Q3$: $<\{(q_3, l_2.d_1, 2), (q_3, l_1.d_1, 4), (q_3, l_2.d_2, 2)\}, \{(q_3, l_2.d_2, 2)\}>$

Assume that the transfers are currently assigned to the processors P_1, P_2, and P_3 in the following way:

- π_1: $< (q_1, l_1.d_1, 2), (q_3, l_1.d_1, 4)>$
- π_2: $< (q_2, l_1.d_2, 3), (q_1, l_2.d_2, 2)>$
- π_3: $< (q_3, l_2.d_1, 2)>$

The present data transfer plans are visualized in Figure 5.

Figure 5
Elimination of conflicts



From the original three sequences, the above transfers are assigned to processes; therefore, we assume that the implementations of the updated sequences are as follows:

- Q_1: $<\{(q_1, l_3.d_1, 3)\}, \{(q_1, l_2.d_1, 3)\}>$
- Q_2: $<\{(q_2, l_2.d_1, 4)\}, \{(q_2, l_3.d_1, 3)\}>$
- Q_3: $<\{(q_3, l_2.d_2, 2)\}, \{(q_3, l_2.d_2, 2)\}>$

Algorithm 2 creates the sets of processes $\boldsymbol{P_{high}} = \{P_1, P_2\}$ and $\boldsymbol{P_{low}} = \{P_3\}$. *Algorithm 3* creates a set of candidate transfer sets $T_c = \{\{(q_1,\ l_3.d_1,\ 3)\},\ \{(q_2,\ l_2.d_1,\ 4)\},\ \{(q_3,\ l_2.d_2,\ 2)\}\}$. Next, the computations of $\boldsymbol{T}(\pi_3) = 2$ and $\boldsymbol{T}(\pi_1) = 6$ contribute to a time frame range $F = (3, 6)$. The data transfers that overlap within F from the tail of plan π_1 and π_2 are saved in $T_a = \{(q_3, l_1.d_1, 4), (q2, l_1.d_2, 3), (q_1, l_2.d_2, 2)\}$.

Next, *Algorithm 4* finds the conflicts. The first set of candidate data transfers $\{(q_1, l_3.d_1, 3)\}$ is removed from T_c because it belongs to the same sequence Q_1 as a data transfer $(q_1, l_2.d_{2,}\ 2)$ and their processing time slots overlap. Next, there is no need to remove from T_c a data transfer $(q_2, l_2.d_1, 4)$ because no conflict is detected for it. A data transfer $(q_3, l_2.d_2, 2)$ is removed from T_c because it accesses the same $l_2.d_2$ as a transfer $(q_1, l_2.d_2, 2)$ from T_a. Finally, *Algorithm 4* returns a set $T_c = \{\{(q_2, l_2.d_1, 4)\}\}$.

4.4.2. *Prioritizing significantly long data transfers*

A strategy for assigning the transfers from the candidate sets *Tc* to the processes is based on the principle of choosing long transfers first. Leaving a long data transfer until the end of a processing plan can result in significant idle time assigned to the processes and an unbalanced workload. Such a situation may arise when all short transfers are assigned to the processes, the workload on each process is approximately the same, and only a few long transfers remain to be assigned. The long transfers then engage only a few processes, leaving the other processes idle and reducing the level of parallelism when conflicts occur between the long transfers. When assigning data transfers to processes, long transfers must be prioritized to ensure efficient workload distribution and to minimize idle time. Another reason for such a strategy is the processing characteristics of queries, which tend to read large data containers first to reduce the amount of data before processing joins. This approach also allows for faster processing by transferring data from lower to higher levels in multi-tiered storage as early as possible.

A strategy for assigning the transfers from the candidate sets T_c to the processes is based on the principle of *choosing long transfers first*. Leaving a long data transfer until the end of a processing plan can result in significant idle time assigned to the processes and an unbalanced workload. Such a situation may arise when all short transfers are assigned to the processes, the workload on each process is approximately the same, and only a few long transfers remain to be assigned. The long transfers then engage only a few processes, leaving the other processes idle and reducing the level of parallelism when conflicts occur between the long transfers. When assigning data transfers to processes, long transfers must be prioritized to ensure efficient workload distribution and to minimize idle time. Another reason for such a strategy is the processing characteristics of queries, which tend to read large data containers first to reduce the amount of data before processing joins. This approach also allows for faster processing by transferring data from lower to higher levels in multi-tiered storage as early as possible.

Algorithm 5 reads a set T_c and detects long data transfers in T_c. Notably, a significantly long data transfer can be the longest in T_c. However, the longest transfer in T_c may not be a significantly long data transfer. This occurs, for example, when the data transfers are long, and the lengths of all data transfers in T_c are approximately the same. In such a case, the longest data transfer in T_c is not significantly longer than the other transfers, and the algorithm does not detect significantly long data transfers in T_c. Data transfers that have approximately the same length do not contribute to a long tail of idle time units. By contrast, if one or more long data transfers in T_c are significantly longer than the others, then they may contribute to a significantly imbalanced workload assigned to the processors.

To calculate the gap percentage G between the longest and other data transfers in T_c *Algorithm 5*, simply utilize Equation (3). Initially, the algorithm determines the length of T_{max}, which represents the longest data transfer. It then proceeds to identify the total number of data transfers n in T_c, excluding those that are the longest. Finally, the algorithm calculates the length of T_{other}, which represents the sum of all other transfers in T_c, except for the most extended data transfers.

$$G = \frac{T_{max} - \left(\frac{T_{other}}{n}\right)}{T_{max}} * 100 \tag{3}$$

Next, a threshold percentage δ is used to find the longest transfers in T_c that are significantly longer than others. For example, suppose that the length of the longest data transfer is 100, and the length of the other transfer is 50. The gap between those two transfers is 50, and if the acceptable threshold value $\delta = 50$, then the longest data transfer is not significantly long. However, if the acceptable threshold value $\delta = 40$, the longest data transfer can be defined as significantly longer than other data transfers. The detailed procedure is shown in *Algorithm 5*.

Algorithm 5: Prioritizing significantly long data transfers (*Rule 1*)

```
Require: A set of candidate data transfer sets $T_c$ and threshold value $\delta$.
Ensure: An updated set of $T_c$.
for each data transfer in $T_c$ do
    Find the length of the longest transfer and save it in $T_{max}$.
Sum the lengths of all transfers in $T_c$ except the length equal to $T_{max}$ and save the result in $T_{other}$.
Use Equation (3) to compute $G$.
if $G \geq \delta$ then
    Eliminate all data transfers from $T_c$ shorter than $T_{max}$.
end if
Return $T_c$.
end for
```

Example 5 Let a threshold value for a long transfer be $\delta = 50$.

Let a set of sets of candidate transfers be $T_c = \{\{(q_1, l_6.d_1, 9), (q_1, l_5.d_1, 3), (q_1, l_3.d_2, 10)\}, \{(q_2, l_3.d_1, 10), (q_2, l_2.d_3, 1)\}\}$.

Therefore, $\boldsymbol{T}_{\max} = 10$ and $\boldsymbol{T}_{\min} = 13$ and $G = 56.67$.

Since G is greater than δ, significant long transfers are detected in T_c. Therefore, T_c must be updated to $\{\{(q_1, l_3.d_2, 10)\}, \{(q_2, l_3.d_1, 10)\}\}$.

4.4.3. *Rule 2: Finding the longest processing sequences*

Rule 2 is used to further reduce the size of T_c. An objective of this rule is to minimize the overall processing time and ensure that the workload is evenly distributed among the various data transfer processes. The rule selects the sets of transfers from T_c that belong to the present longest sequences in $\boldsymbol{Q}$. In order to determine the total length of each sequence in $\boldsymbol{Q}$, it is not sufficient to simply add the τ values from each sequence. This is because the sets of data transfers from each sequence can be processed in parallel over multiple processes, but each set must be processed sequentially.

To estimate the time required to process each set in parallel, it is necessary to find the average processing time by summing all the τ values from each set and dividing this sum by the number of processes. In Equation (4), let $Avg(T_i)$ denote the average processing time for a set of data transfers T_i, p denotes the number of processes, and T_i represents the set of data transfers. The detailed calculation is presented in Equation (4) below.

$$Avg(T_i) = \frac{1}{p}\sum\nolimits_{i=n}^{m} \tau_n \tag{4}$$

Once the average processing time for each set is obtained, it is possible to sum all the average times to determine the total processing time for a single sequence. This leads to an accurate estimation of the length of the sequence. In Equation (5), $T(Q_i)$ is denoted as the estimated length for a sequence Q_i. The detailed calculation is presented in Equation (5) below.

$$T(Q_i) = \sum\nolimits_{n=i}^{j} Avg(T_n) \tag{5}$$

Algorithm 6 uses a set of sequences $\boldsymbol{Q}$ and a set of candidate transfer sets T_c from *Rule 1*. It applies Equations (4) and (5) to compute the length of each sequence. Then it selects sequences with the lengths equal to the maximum T value and retains only their data transfer sets, removing any data transfer sets violating *Rule 2*. *Algorithm 6* returns an updated T_c.

Algorithm 6: Finding the longest processing sequences (*Rule 2*)

```
Require: A set of sequences $\boldsymbol{Q} = \{Q_1, \ldots, Q_m\}$ and a set of candidate transfers sets $T_c$ from Rule 1.
Ensure: An updated set of candidate transfers sets $T_c$.
for each sequence in $\boldsymbol{Q}$ do
    Let the current sequence be $Q_i$.
    Use Equations (4) and (5) to compute the length $T(Q_i)$$.
end for
Let $T_{max}$ be the largest value in $T(Q_1), \ldots, T(Q_m)\}$.
for each sequence in $\boldsymbol{Q}$ do
    Let the current sequence be $Q_i$.
    if $T(Q_i) \neq T_{max}$ then
        Remove a set of candidate transfers from $T_c$ that belongs to a sequence $Q_i$.
    end if
end for
Return an updated set of candidate data transfer sets $T_c$.
```

Example 6 Consider the sequences Q_1, Q_2, and Q_3 such that $T(Q_1) = 50$, $T(Q_2) = 30$, and $T(Q_3) = 50$. Let a set of candidate data transfer sets be $T_c = \{\{(q_1, l_1.d_1, 2), (q_1, l_2.d_1, 3)\}, \{(q_2, l_1.d_2, 3), (q_2, l_1.d_3, 4)\}, \{(q_3, l_2.d_2, 4)\}\}$. The length of the longest sequence is $T(Q_1) = 50$. The length of sequence Q_3 is the same as the length of sequence Q_1. According to *Rule 2*, we remove a set of candidate data transfers belonging to Q_2 because $T(Q_2)$ is not equal to $T(Q_1)$. An updated $T_c = \{\{(q_1, l_1.d_1, 2), (q_1, l_2.d_1, 3)\}, \{(q_3, l_2.d_2, 4)\}\}$.

4.4.4. Rule 3: Minimizing the future conflicts

When scheduling data transfers, it is important to consider the likelihood of future conflicts. *Rule 3* eliminates data transfers from T_c that are less likely to cause conflicts in the future. The rule aims to eliminate future conflicts by prioritizing data transfers that are likely to cause them. Fewer conflicts in turn lead to less idle time assigned to processes and provide a well-balanced workload. However, identifying future conflicts through analysis of entire sequences in $\boldsymbol{Q}$ might take too long and the sets that fall behind the sequences may not contribute to future conflicts involving transfers in T_c. Therefore, prediction of future conflicts requires analysis within the limited number of data transfer sets from each sequence. The algorithm, hence, only uses a certain number of sets, denoted by f, from each sequence to discover future conflicts.

For instance, if a transfer in T_c is found to have a high chance of conflict with other transfers not included in T_c, it will be given priority and allocated in the current stage. Any remaining transfers in T_c that have a low chance for future conflicts are removed from T_c, while those with a high likelihood are retained in T_c. *Algorithm 7* eliminates data transfers from T_c using *Rule 3*.

The algorithm processes a set of sequences $\boldsymbol{Q}$ a sequence of tiers $\boldsymbol{L}$ and f sets in each sequence, and it outputs an updated set of candidate data transfers T_c. The algorithm skips the first set from each sequence, selects f sets from each sequence, and then adds them to a temporary set of data transfers, T_{temp}.

Next, the algorithm creates a matrix M of future conflicts. This matrix has several rows m equal to the total number of sequences in $\boldsymbol{Q}$ and a number of columns n equal to the number of devices in $\boldsymbol{L}$. Each column in the matrix corresponds to a device from a tier $l \in \boldsymbol{L}$.

The algorithm iterates over T_{temp} and considers the current data transfer as $(q_i, l_j.d_k, \tau_p)$. Let row x be associated with sequence Q_i and column y be associated with device $l_j.d_k$. The algorithm increases a value at an entry $M(x, y)$ by 1. Once the matrix is filled with values, the algorithm calculates the values in a vector s. In particular, for all sequences in $\boldsymbol{Q}$, the algorithm computes the total number of transfers that utilize each device $l.d$ and saves it in a vector $s(l.d)$.

For instance, suppose that two data transfers access the same device $l_i.d_j$ in Q_i and one transfer accesses the same device $l_i.d_j$ in Q_j. The total number of times the device $l_i.d_j$ is accessed is 3 and a value in a vector $s(l_i.d_j)$ is thus set to 3. A vector s is used to determine the busiest devices once all the data transfers from $\boldsymbol{Q}$ have been processed.

In Equation (6), fill up the vector value $s(l.d)$ by summing all the value from column $l.d$. Equation (6) shows how the algorithm fills the entries in a vector s.

$$s(l.d) = \sum_{i=1}^{n} M(Q_i,\ l.d) \tag{6}$$

Next, the algorithm calculates the values in a vector c. Notably, many transfers accessing a device do not necessarily mean the device will contribute to future conflicts but rather that the device is busier than the other devices. If multiple data transfers from the same sequence access the same device, they can be processed sequentially without causing conflicts. Thus, even if the device appears busy, it may not necessarily cause conflicts. To predict potential conflicts, the algorithm utilizes Equation (7) to compute the values in a vector c. This vector indicates how many sequences of data transfers access the same device on a ratio basis. The values in c range from 0 to 1. The total number of rows in the matrix is n. Determine the number of rows containing a zero in each column, and save the result for each columns as $z(l.d)$. For instance, get the number of zero values in column $l.d$ and save the result in $z(l.d)$. The total number of rows in the matrix is denoted as m and Equation (7) below to calculate the conflict ratio $c(l.d)$ for column $l.d$.

$$c(l.d) = 1 - \frac{z(l.d)}{m - 1} \tag{7}$$

Merely considering the value of c is not enough to determine whether a transfer will cause conflict when accessing a busy device. This is because two or more devices could have the same c value but different data transfer distributions. The algorithm therefore adds the vectors $c(l.d)$ and $s(l.d)$ together to estimate future conflicts more efficiently. Equation (8) is used to compute the values in a vector $S(l.d)$ that estimate the chances of future conflicts by summing vector values $s(l.d)$ and $c(l.d)$.

$$S(l.d) = s(l.d) + c(l.d) \tag{8}$$

Next, the algorithm chooses the devices that offer the largest value of S. Thereafter, it picks the data transfers from T_c that are accessing the selected devices, and it excludes the remaining data transfers from T_c that are not accessing the selected devices.

Algorithm 7: Minimizing the total number of future conflicts (*Rule 3*)

```
Require: A set of sequences $\boldsymbol{Q} = \{Q_1, \ldots, Q_m\}$, a sequence of tier
$\boldsymbol{L} = <l_0, \ldots, l_n>$, a set of candidate data transfer sets $T_c = \{\{(q_i, l_j.d_k, \tau_k), \ldots, (q_j, l_k.d_v, \tau_i)\}, \ldots, \{(q_k, l_i.d_n, \tau_j), \ldots, (q_l, l_n.d_i, \tau_n)\}\}$,
and the $f$ sets for further analysis of each sequence.
Ensure: An updated set of candidate transfers sets $T_c$.
Create a temporary empty set $T_{temp}$.
for each sequence in $\boldsymbol{Q}$ do
    Let the current sequence be $Q_i$.
    Select $f$ sets from $Q_i$ – first set.
    Append selected sets of data transfers to $T_{temp}$.
end for
Create a matrix $M(m \times n)$, where $m$ represents the total
number of sequences in $\boldsymbol{Q}$ and $n$ represents the total number of
devices associated with each tier in $\boldsymbol{L}$.
for each data transfer in $T_{temp}$ do
    Let the current data transfer be $(q_i, l_j.d_k, \tau_p)$.
    Let the location of $Q_i$ be in row $x$ and the location of device
        $l_j.d_k$ be in column $y$.
    Increase the value at $M(x, y)$ by 1.
end for
for $i: = 1$ to $n$ do
    Let the current column be $y$ and it is associated with device
        $l_x d_y$.
    Compute the vector value $s(l_x d_y)$ using Equation (6).
    Count how many zeros appear in the current column and set
        the value to $z(l_x d_y)$.
    Compute the vector value $c(l_x d_y)$ using Equation (7).
    Compute the vector value $\boldsymbol{S}(l_x d_y)$ using Equation (8).
end for
Sort the vector $\boldsymbol{S}$ in descending order.
Let the first vector value be associated with $\boldsymbol{S}(l_i.d_j)$.
Select all the data transfers from $T_c$ that are accessing the
device $l_i.d_j$.
if data transfers are selected then
    Keep all the selected data transfers in $T_c$ and remove the un-
        selected data transfers from $T_c$.
    Exit
end if
Return an updated set of candidate data transfer sets $T_c$.
```

Example 7 We consider a multi-tiered persistent storage with four tiers $\boldsymbol{L} = < l_1, l_2, l_3, l_4>$.

A tier l_1 is implemented as a set of devices $\boldsymbol{D}_1 = \{l_1.d_1, l_1.d_2\}$.
A tier l_2 is implemented as a set of devices $\boldsymbol{D}_2 = \{l_2.d_1\}$.
A tier l_3 is implemented as a set of devices $\boldsymbol{D}_3 = \{l_3.d_1, l_3.d_2\}$.
A tier l_4 is implemented as a set of devices $\boldsymbol{D}_4 = \{l_4.d_1\}$.

We assume the following present state of the sequences $\boldsymbol{Q} = \{Q_1, Q_2, Q_3\}$.

- $Q_1 = <\{ (q_1, l_0.d_1, 3), (q_1, l_0.d_2, 2)\}, \{(q_1, l_3.d_3, 3)\}, \{(q_1, l_4.d_1, 2)\}, \ldots >$,
- $Q_2 = <\{ (q_2, l_1.d_2, 4), (q_2, l_1.d_1, 3)\}, \{(q_2, l_1.d_1, 4), (q_2, l_2.d_1, 2)\}, \{(q_2, l_3.d_2, 2), (q_2, l_3.d_1, 4)\}, \ldots >$,
- $Q_3 = <\{(q_3, l_0.d_3, 2)\}, \{(q_3, l_1.d_2, 5), (q_3, l_4.d_1, 2), (q_3, l_1.d_1, 2)\}, \{(q_3, l_1.d_2, 3), (q_3, l_3.d_1, 2)\}, \ldots>$.

The current $T_c = \{\{(q_1, l_0.d_1, 3), (q_1, l_0.d_2, 2)\}, \{(q_2, l_1.d_2, 4), (q_2, l_1.d_1, 3)\}, \{(q_3, l_0.d_3, 2)\}\}$. We assume the maximum number of sets for future conflict analysis $f = 2$. Therefore, a temporary set of data transfers is $T_{temp} = \{(q_1, l_3.d_3, 3), (q_1, l_4.d_1, 2), (q_2, l_1.d_1, 4), (q_2, l_2.d_1, 2), (q_2, l_3.d_2, 2), (q_2, l_3.d_1, 4), (q_3, l_1.d_2, 5), (q_3, l_4.d_1, 2), (q_3, l_1.d_1, 2), (q_3, l_1.d_2, 3), (q_3, l_3.d_1, 2)\}$. *Algorithm 7* creates a conflict matrix based on the sequences in T_{temp} and a matrix of conflicts with vectors *s, c,* and ***S*** as shown in Table 1.

Table 1
A matrix of conflicts *M* with the vectors *s, c,* and *S*

	$l_1.d_1$	$l_1.d_2$	$l_2.d_1$	$l_3.d_1$	$l_3.d_2$	$l_3.d_3$	$l_4.d_1$
Q_1	0	0	0	0	0	1	1
Q_2	1	0	1	1	1	0	0
Q_3	2	2	0	1	0	0	1
s	3	2	1	2	1	1	2
c	0.50	0.00	0.00	0.50	0.00	0.00	0.50
$\boldsymbol{S}$	3.50	2	1	2.5	1	1	2.5

Since the largest value of ***S*** is 3.50, associated with column $l_1.d_1$.Therefore, the algorithm picks only data transfers accessing the device $l_1.d_1$ and removes the rest of the sets from T_c. The updated T_c will be $\{\{(q_2, l_1.d_2, 4), (q_2, l_1.d_1, 3)\}\}$.

4.4.5. Rule 4: Finding the largest number of data transfers

The scheduling *Rule 4* selects sets of data transfers from T_c that belong to sequences with the largest number of data transfers. The sequences with the largest number of data transfers can contain several small data transfers. Smaller data transfers can produce fewer chances of future conflict and provide well-balanced allocation over processes. The scheduling rule balances data transfer allocation in the process and reduces the idle time during the allocation.

Algorithm 8 takes a set of sequences Q and a set of candidate transfer sets T_c from *Rule 3*. Then, it counts the total number of data transfers $C(Q_i)$ in each sequence Q_i in Q. Thereafter, the algorithm selects sequences with total transfers equal to $C(Q_j)$ and subsequently removes all sets of data transfers that do not belong to the selected sequences from T_c. Finally, the updated T_c is returned.

Algorithm 8: Finding the largest number of data transfers (*Rule 4*)

```
Require: A set of sequences $\boldsymbol{Q} = \{Q_1, \ldots, Q_m\}$, and a set of candidate transfers sets $T_c$ from Rule 3.
Ensure: An updated set of candidate transfers sets $T_c$.
for each sequence in $\boldsymbol{Q}$ do
    Let the current sequence be $Q_i$.
    Count the number of data transfers in $Q_i$ and let $\boldsymbol{C}(Q_i)$ store the count result of $Q_i$.
end for
Let a sequence $Q_j$ consists of the largest number of data transfers $\boldsymbol{C}(Q_j)$.
for each sequence in $\boldsymbol{Q}$ do
    Let the current sequence be $Q_i$.
    if $\boldsymbol{C}(Q_i) \neq \boldsymbol{C}(Q_j)$ then
        Remove a set of candidate transfers from $T_c$ that belongs to a sequence $Q_i$.
    end if
end for
Return an updated set of candidate data transfer sets $T_c$.
```

Example 8 We consider a set of sequences $Q = \{Q_1, Q_2, Q_3\}$, and we assume the length of each sequence $C(Q_1) = 12$, $C(Q_2) = 12$, and $C(Q_3) = 8$. A set of candidate data transfers is $T_c = \{\{(q_1, l_1.p_1, 2), (q_1, l_2.p_1, 3)\}, \{(q_2, l_2.p_2, 4)\}, \{(q_3, l_3.p_1, 2), (q_3, l_3.p_2, 2)\}\}$. The longest sequences are Q_1 and Q_2 due to $C(Q_1) = C(Q_2) = 12$. According to *Rule 4*, we need to remove a set of candidate data transfers belonging to Q_3 because $C(Q_3)$ is not equal to $C(Q_1)$. Final updated result is $T_c = \{\{(q_1, l_1.p_1, 2), (q_1, l_2.p_1, 3)\}, \{(q_2, l_2.p_2, 4)\}\}$.

4.4.6. Rule 5: Random selection of data transfers

When multiple sets of data transfers are returned by *Rule 4* in T_c, the scheduling rule called *Rule 5* is applied to randomly pick one of the sets. Due to the application of multiple rules that decrease the size of T_c, the remaining data transfer sets will effectively minimize the idle time in future allocation while also balancing the workload across processes. Thus, assigning any data transfers from T_c to lower workload processes could be beneficial. In *Algorithm 9*, a set of candidate data transfers T_c is used, and *Rule 5* is implemented to select one set for allocation on processes randomly.

Algorithm 9: Random (*Rule 5*)

```
Require: A set of candidate transfers sets $T_c$ from Algorithm 8.
Ensure: An updated set of candidate transfers sets $T_c$.
Select a set of candidate data transfers randomly from $T_c$.
Remove all other sets of candidate data transfers from $T_c$ except the one that was selected.
Return an updated set of candidate data transfer sets $T_c$.
```

4.4.7. Rule 6: Longest data transfer and process with shortest workload

To balance workload, the scheduling *Rule 6* is implemented in this section. This rule selects a transfer candidate from T_c that is larger in size than the others and assigns it to a process with the lowest workload, thereby guaranteeing that workload is distributed evenly among the processes.

Algorithm 10 implements Rule 6 by taking a set of candidate data transfers T_c returned from *Algorithm 9*, a set of data transfer processes denoted as $\boldsymbol{P} = \{P_1, \ldots, P_j\}$ and a set of process plans denoted as $\{\pi_1, \ldots, \pi_j\}$. The algorithm sorts all data transfers from T_c in descending order based on their tau values before iterating over T_c and selecting the current data transfer $(q_i, l_j.d_k, \tau_n)$. Thereafter, the algorithm computes the workload of the process using Equation (1) and then assigns the current data transfer $(q_i, l_j.d_k, \tau_n)$ to the process plan with the lowest workload, denoted as π_j. This iteration continues until all data transfers from T_c are assigned to the process plans associated with the processes in $\boldsymbol{P}$. Finally, *Algorithm 10* provides an updated set of process plans $\{\pi_1, \ldots, \pi_j\}$.

Algorithm 10: Longest data transfer and process with shortest workload (*Rule 6*)

```
Require: A set of candidate transfers sets $T_c$ from Algorithm 9, a set of processes $\boldsymbol{P} = \{P_1, \ldots, P_j\}$, and a set of process plans $\{\pi_1, \ldots, \pi_j\}$.
Ensure: An updated set of process plans $\{\pi_1, \ldots, \pi_j\}$.
Arrange all data transfers in $T_c$ in descending order according to their $\tau$ value.
for each data transfer in $T_c$ do
    Let the current data transfer be $(q_i, l_j.d_k, \tau_n)$.
    Computes the workload for each process by using Equation (1).
    Select a process plan $\pi_j$, which is the shortest workload among all process plans.
    Assign a data transfer $(q_i, l_j.d_k, \tau_n)$ to a process plan $\pi_j$.
end for
Return the updated version of a set of process plans $\{\pi_1, \ldots, \pi_j\}$.
```

Example 9 In this example, we have a set of data transfers denoted as T_c. It consists of three transfers: $(q_i, l_j.d_k, 5)$, $(q_i, l_i.d_j, 3)$, and $(q_i, l_k.d_i, 6)$.

First, algorithm needs to arrange the values of T_c in descending order using τ as follows $<(q_i, l_i.d_j, 3), (q_i, l_j.d_k, 5), (q_i, l_k.d_i, 6)>$. Next, algorithm iterates over T_c and the current data transfer is $(q_i, l_i.d_j, 3)$. Next, the algorithm calculates the workload for each process using Equation (1). The results are $\boldsymbol{T}(\pi_1) = 5$ and $\boldsymbol{T}(\pi_2) = 2$. Since π_2 has the lowest workload in this iteration, the algorithm assigns $(q_i, l_i.d_j, 3)$ to a plan $\pi_2 = <(q_k, l_i.d_j, 3), (q_i, l_i.d_j, 3)>$. After completing the iteration, the algorithm returns the updated process plan as follows:

- $\pi_1 = < (q_i, l_i.d_n, 2), (q_j, l_k.d_m, 3), (q_i, l_j.d_k, 5)>$
- $\pi_2 = <(q_k, l_i.d_j, 3), (q_i, l_i.d_j, 3), (q_i, l_k.d_i, 6)>$.

5. Experiments

This study conducted various experiments to evaluate the proposed rules and their impact on different processes. Multiple cases were created and implemented in different categories. Some experiments focused on sequences with similar lengths, where transfers within those queries exhibited noticeable variations in length. This was a strict test for *Rule 1*. Other experiments maintained consistent transfer lengths within sequences of similar lengths to avoid invoking *Rule 1*. Furthermore, to test *Rule 3*, certain experiments featured sequences with significant disparities in length while keeping transfer lengths consistent. A subset of experiments introduced significantly longer sequences and transfers to examine the combined effects of *Rule 1*, *Rule 3*, and *Rule 4*. Specific experiments intentionally introduced conflicts into sequences with similar lengths to assess the efficacy of *Rule 0*. By contrast, another set of experiments prolonged certain sequences and introduced multiple conflicts to evaluate the interplay of *Rule 0*, *Rule 1*, and *Rule 2*. Most experiments involved sequences of similar length without conflicts, facilitating the evaluation of random allocation using *Rule 5* and the comprehensive assessment of all rules in a combined fashion. A distinct subset featured significantly longer sequences without conflicts, providing a unique perspective on the applicability of all rules collectively. This comprehensive approach allowed for a thorough investigation of the rules' effectiveness across a wide spectrum of conditions.

Eight experiments were conducted on a stimulated multi-tiered persistent storage to investigate various aspects of data transfer allocation strategies in different query scenarios and process configurations. In our experiments, we stimulated multi-tiered persistent use, with four tiers denoted as $\boldsymbol{L} = <l_0, l_1, l_2, l_3>$. However, due to practical limitations, it was not feasible to physically test all devices in a short period of time for a large database with complete tiers. Therefore, instead of using realistic devices, we analyzed the available devices from the market and incorporated their read/write speeds into a simulator. Each tier has its own set of device configurations, which are unique and distinct.

- Tier 0 (l_0): Consists of three devices, namely $\{d_1, d_2, d_3\}$, all equipped with HDD storage.
- Tier 1 (l_1): Comprises three devices, $\{d_1, d_2, d_3\}$, each utilizing SSD storage technology.
- Tier 2 (l_2): Comprises three devices, $\{d_1, d_2, d_3\}$, all employing high-speed SSD storage.
- Tier 3 (l_3): Consists of one device, specifically $\{d_1\}$, each featuring an NVMe drive.

The read and write speeds associated with each of these tiers are depicted in Figure 6, providing essential information for the experiment's evaluation and resource allocation considerations.

5.1. Experiment configuration

Moreover, we used the TPC-H benchmark database [20] as the foundation for our experiment's database and query configuration. The TPC-H database was scaled down to 10GB to facilitate efficient testing within a manageable range. Data were distributed across storage tiers, as detailed in Table 2, to explore multi-tiered persistent data distribution. From 22 available queries in TPC [20], we selected 15 to investigate. The experiment configuration comprised eight distinct experiments. Each is designed to explore various aspects of our multi-tiered data persistence system. Experiment 1 focused on five sequences, namely $\{Q_1, Q_2, Q_3, Q_4, Q_5\}$, and employed two processes, evaluating all four scheduling methods for transfer allocation plans. Experiment 2, similarly to Experiment 1, used the same five queries $\{Q_1, Q_2, Q_3, Q_4, Q_5\}$, but employed three processes and assessed all four scheduling methods for transfer allocation plans. In Experiment 3, five queries, namely $\{Q_6, Q_7, Q_8, Q_9, Q_{10}\}$, were utilized, and two processes were employed, with all four scheduling methods applied for transfer allocation plans. Experiment 4, similarly to Experiment 3, used vie queries, namely $\{Q_6, Q_7, Q_8, Q_9, Q_{10}\}$, but employed three processes and assessed all four scheduling methods for transfer allocation plans. Experiment 5 focused on five queries, namely $\{Q_{11}, Q_{12}, Q_{13}, Q_{14}, Q_{15}\}$, and employed two processes, evaluating all four scheduling methods for transfer allocation plans. Similarly, to Experiment 5, Experiment 6 used the same five queries, namely $\{Q_{11}, Q_{12}, Q_{13}, Q_{14}, Q_{15}\}$, but it employed three processes and assessed all four scheduling methods for transfer allocation plans. Experiment 7 focused on seven queries, namely $\{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7\}$, and three processes were employed, with all four scheduling methods applied for transfer allocation plans. The final experiment utilized eight queries, namely $\{Q_8, Q_9, Q_{10}, Q_{11}, Q_{12}, Q_{13}, Q_{14}, Q_{15}\}$, and employed three processes, evaluating all four scheduling methods for transfer allocation plans.

Table 2
Table locations and estimated size

Table name	Location	Number of rows	Size (in MB)	Each row (in bytes)
LINEITEM	$l_1.d_1$	24,000,000	2,563	112
LINEITEM	$l_1.d_2$	24,000,000	2,563	112
LINEITEM	$l_1.d_2$	24,000,000	2,563	112
ORDERS	$l_1.d_3$	10,000,000	992	104
ORDERS	$l_1.d_4$	5,000,000	495	104
CUSTOMER	$l_0.d_3$	1,500,000	256	179
NATION	$l_0.d_1$	250	< 1	128
REGION	$l_0.d_1$	50	< 1	124
PART	$l_0.d_1$	2,000,000	296	155
PARTSUPP	$l_0.d_2$	8,000,000	1,099	144
SUPPLIER	$l_0.d_2$	100,000	15	159
		Total	9,067	

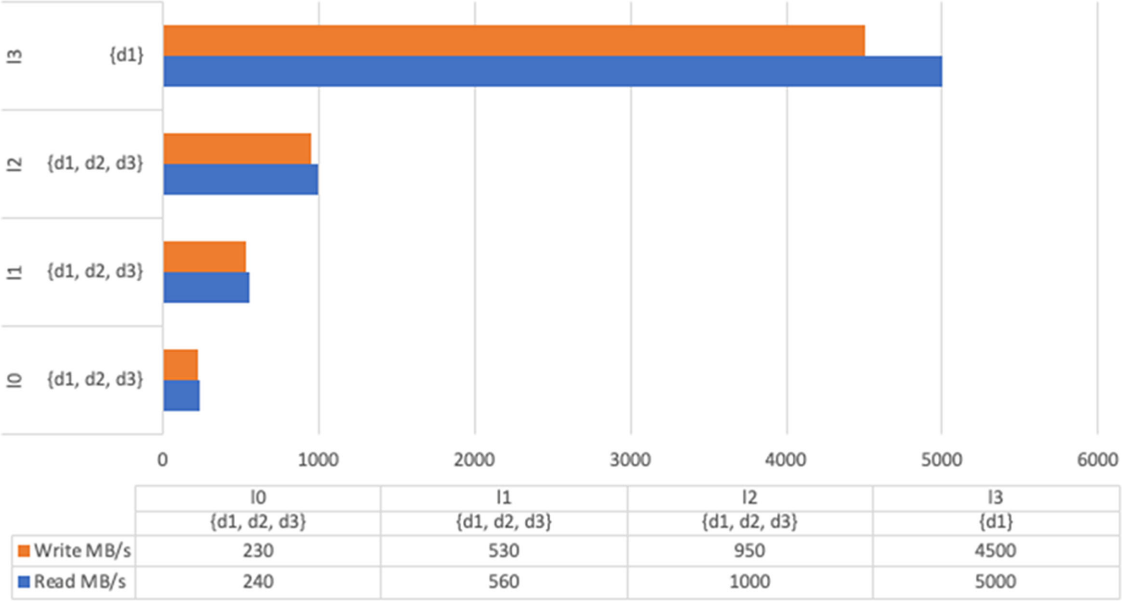
From 22 available queries in TPC [20], we selected 15 to investigate. The experiment configuration comprised eight distinct experiments. Each is designed to explore various aspects of our multi-tiered data persistence system. Experiment 1 focused on five sequences, namely $\{Q_1, Q_2, Q_3, Q_4, Q_5\}$, and employed two processes, evaluating all four scheduling methods for transfer allocation plans. Experiment 2, similarly to Experiment 1, used the same five queries $\{Q_1, Q_2, Q_3, Q_4, Q_5\}$, but employed three processes and assessed all four scheduling methods for transfer allocation plans. In Experiment 3, five queries, namely $\{Q_6, Q_7, Q_8, Q_9, Q_{10}\}$, were utilized, and two processes were employed, with all four scheduling methods applied for transfer allocation plans. Experiment 4, similarly to Experiment 3, used vie queries, namely $\{Q_6, Q_7, Q_8, Q_9, Q_{10}\}$, but employed three processes and assessed all four scheduling methods for transfer allocation plans. Experiment 5 focused on five queries, namely $\{Q_{11}, Q_{12}, Q_{13}, Q_{14}, Q_{15}\}$, and employed two processes, evaluating all four scheduling methods for transfer allocation plans. Similarly, to Experiment 5, Experiment 6 used the same five queries, namely $\{Q_{11}, Q_{12}, Q_{13}, Q_{14}, Q_{15}\}$, but it employed three processes and assessed all four scheduling methods for transfer allocation plans. Experiment 7 focused on seven queries, namely $\{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7\}$, and three processes were employed, with all four scheduling methods applied for transfer allocation plans. The final experiment utilized eight queries, namely $\{Q_8, Q_9, Q_{10}, Q_{11}, Q_{12}, Q_{13}, Q_{14}, Q_{15}\}$, and employed three processes, evaluating all four scheduling methods for transfer allocation plans.

For experiments, we use stimulated multi-tiered persistent storage. Therefore, we need to process data transfer plans for queries on that simulator. To achieve this, we first need to convert queries into EPNs and then convert them into data transfer plans, which are sequences of sets of data transfers.

To accomplish this, we use MySQL software to transform each query into a query explain plan (a query processing graph). This graph provides the data flow and estimated sizes of input and output data for each operation. The example of the graph generated by MySQL is shown in Figure 2.

From this graph, we can easily transform it into EPN and generate data transfer plans based on the availability of the number of data transfer processes. Since the number of data transfer processes used

Figure 6
A sample data transfer speed for the tier with a set of devices



	l0 {d1, d2, d3}	l1 {d1, d2, d3}	l2 {d1, d2, d3}	l3 {d1}
🟧 Write MB/s	230	530	950	4500
🟦 Read MB/s	240	560	1000	5000

in each experiment is different, generating data transfer plans for each experiment will also be different. There are several ways to generate data transfer plans, and processing on that plan proceeds different processing time too. Different scheduling methods show different ways of generating the data transfer plans.

The experiments used four distinct scheduling methods to generate data transfer plans. Four different scheduling methods were evaluated during each experiment, each with a unique way of allocating resources. The first method, optimal scheduling, aimed to create the most efficient allocation plan possible. The second method, known as rules-based scheduling or Algorithm 1, was proposed in the study. The third method, allocating long transfers first, prioritized the completion of long data transfers. The fourth method allocates long sequences first, prioritized data transfers from long sequences. These four methods were used to evaluate the effectiveness and performance of various resource allocation strategies in the experiments.

Table 3
Experiment results

Experiment	Optimal scheduling	Rule-based scheduling	Long data transfer first	Long sequence first
Experiment 1	54	55	57	56
Experiment 2	36	42	42	42
Experiment 3	51	51	51	51
Experiment 4	34	34	35	35
Experiment 5	30	30	31	30
Experiment 6	21	21	21	21
Experiment 7	43	43	49	44
Experiment 8	47	47	49	48
Total	316	323	335	327

5.2. Experiment result

A detailed analysis of the experiment result of the eight different experiments, each evaluating various strategies for data transfer allocation. The results from these experiments are summarized in Table 3.

5.2.1. Rule-based and optimal strategies

Across most experiments (Experiments 1–6, 8), the *rule-based* and *optimal* strategies demonstrated comparable execution times. This intriguing finding suggests that the *rule-based* approach closely approximates the theoretically *optimal* strategy.

5.2.2. Long transfer and long sequence strategies

Notably, strategies prioritizing long transfers or long sequences occasionally resulted in extended execution times compared to *rule-based* and *optimal* approaches. This intriguing observation underscores the importance of careful consideration in resource allocation decision-making processes, highlighting the potential trade-offs associated with prioritizing specific data transfer characteristics.

Specifically, the *rule-based* scheduling approach emerged as a promising contender, often showcasing performance on par with the optimal strategy, with a total time unit of 312. In contrast, the long transfer first strategy consistently demanded more time (total time unit of 335), potentially due to its emphasis on prioritizing long transfer tasks without necessarily optimizing overall efficiency. While the long sequence first strategy exhibited relatively better performance with a total time unit of 327, it still trailed behind optimal and rule-based strategies, thereby underscoring the nuanced dynamics at play in data transfer optimization.

In conclusion, the *rule-based* scheduling approach emerges as a robust and effective strategy for minimizing data transfer time across a diverse array of scenarios. Despite potential requirements for manual rule definition, the *rule-based* approach offers commendable efficiency and competitiveness when compared to the optimal strategy. However, it is imperative to acknowledge the potential drawbacks associated with strategies prioritizing long transfers or sequences, as they may inadvertently lead to suboptimal outcomes. Hence, careful consideration and evaluation of various scheduling methods are crucial for informed resource allocation decisions in multi-tiered persistent storage environments.

6. Summary, Conclusions, and Future Work

This paper introduces scheduling algorithms to enhance parallel data transfers in multi-tiered persistent storage systems. These algorithms are meticulously crafted, considering query sets, storage tier sequences, data transfer sequences, and predefined thresholds for identifying prolonged data transfers.

Algorithm 1 is the main algorithm, coordinating other algorithms for crafting parallel data transfer plans for individual processes. It initiates by selecting potential data transfers from a sequence of data transfer sets, storing them in a set awaiting assignment to processes. Subsequent algorithms refine these sets, resolving conflicts and streamlining transfers, ultimately striving to condense them to a single set allocated to processors with minimal workload.

Experimental results confirm that the generated parallel data transfer plans exhibit near-optimal performance, with some instances achieving perfection. The proposed algorithms accommodate varying query characteristics, yet occasional suboptimal plans may arise due to idle time units in specific scenarios, highlighting areas for further refinement.

In conclusion, this study presents an efficient approach to scheduling parallel data transfers in complex storage environments. The algorithms reduce candidate transfer sets through elimination rules, albeit with varying computational demands.

Moving forward, several challenges warrant attention. First, the scheduling algorithms outlined in the paper sometimes yielded suboptimal solutions, leaving several idle time units in the final plans. It would be useful to further optimize the plans that have already been generated. Second, deciding which queries to prioritize for processing presents an interesting challenge: not all queries need to be processed at the highest tiers of persistent storage. So, prioritization could add flexibility to the use of said storage. The third issue is the preprocessing of an input set of queries, which could lead to partial optimization of the queries before the scheduling algorithms are applied.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Author Contribution Statement

Nan Noon Noon: Conceptualization, Methodology, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization, Project administration. **Janusz Getta and Tianbing Xia:** Conceptualization, Methodology, Formal analysis, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration.

References

[1] Kougkas, A., Devarajan, H., & Sun, X. H. (2018). Hermes: A heterogeneous-aware multi-tiered distributed I/O buffering system. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, 219–230. https://doi.org/10.1145/3208040.3208059

[2] Junaid, M., Shaikh, A., Hassan, M. U., Alghamdi, A., Rajab, K., Al Reshan, M. S., & Alkinani, M. (2021). Smart agriculture cloud using AI based techniques. *Energies*, *14*(16), 5129. https://doi.org/10.3390/en14165129

[3] Chang, J., Shao, B., Ji, Y., & Bian, G. (2020). Efficient identity-based provable multi-copy data possession in multi-cloud storage, revisited. *IEEE Communications Letters*, *24*(12), 2723–2727. https://doi.org/10.1109/LCOMM.2020.3013280

[4] Ignite. (n.d.). *Multi-tier storage*. Retrieved from: https://ignite.apache.org/arch/multi-tier-storage.html

[5] Noon, N. N., Getta, J. R., & Xia, T. (2022). Scheduling parallel data transfers in multi-tiered persistent storage. In *Recent Challenges in Intelligent Information and Database Systems: 14th Asian Conference*, 437–449. https://doi.org/10.1007/978-981-19-8234-7_34

[6] Spiceworks. (n.d.). *Data storage trends in 2020 and beyond*. Retrieved from: https://www.spiceworks.com/sw-marketing/reports/storage-trends-in-2020-and-beyond/

[7] Myung, K., Kim, S., Yeom, H. Y., & Park, J. (2021). Efficient and scalable external sort framework for NVMe SSD. *IEEE Transactions on Computers*, *70*(12), 2211–2217. https://doi.org/10.1109/TC.2020.3041220

[8] Peng, B., Yang, M., Yao, J., & Guan, H. (2021). A throughput-oriented NVMe storage virtualization with workload aware management. *IEEE Transactions on Computers*, *70*(12), 2112–2124. https://doi.org/10.1109/TC.2020.3037817

[9] Yang, Z., Hoseinzadeh, M., Andrews, A., Mayers, C., Evans, D. T., Bolt, R. T., . . . , & Swanson, S. (2017). AutoTiering: Automatic data placement manager in multi-tier all-flash datacenter. In *IEEE 36th International Performance Computing and Communications Conference,* 1–8. https://doi.org/10.1109/PCCC.2017.8280433

[10] Jinyong, C., Bilin, S., Yanyan, J., & Genqing, B. (2020). Efficient identity based provable multi-copy data possession in multi-cloud storage, revisited. *IEEE Communications Letters*, *24*(12), 2723–2727. https://doi.org/10.1109/TCC.2019.2929045

[11] Rudek, R. (2017). Scheduling on parallel processors with varying processing times. *Computers & Operations Research*, *81*, 90–101. https://doi.org/10.1016/j.cor.2016.12.007

[12] Werner, F., Burtseva, L., & Sotskov, Y. N. (2018). Special issue on algorithms for scheduling problems. *Algorithms*, *11*(6), 87. https://doi.org/10.3390/a11060087

[13] Noon, N. N., Getta, J. R., & Xia, T. (2021). Optimization query processing for multi-tiered persistent storage. In *IEEE 4th International Conference on Computer and Communication Engineering Technology*, 131–135. https://doi.org/10.1109/CCET52649.2021.9544285

[14] Stickler, K. C. H. W. (2023). *LP-based approximation algorithms for partial-ordered scheduling and matroid augmentation*. Master's Thesis, London School of Economics and Political Science.

[15] Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., Sterna, M., & Weglarz, J. (2019). *Handbook on scheduling: From theory to*

practice. Switzerland: Springer. https://doi.org/10.1007/978-3-319-99849-7

[16] Bosman, T., Frascaria, D., Olver, N., Sitters, R., & Stougie, L. (2019). Fixed-order scheduling on parallel machines. In *Integer Programming and Combinatorial Optimization: 20th International Conference*, 88–100. https://doi.org/10.1007/978-3-030-17953-3_7

[17] Lofstead, J., Dayal, J., Jimenez, I., & Maltzahn, C. (2013). Efficient transactions for parallel data movement. In *Proceedings of the 8th Parallel Data Storage Workshop*, 1–6. https://doi.org/10.1145/2538542.2538567

[18] Deng, C., Li, G., Zhou, Q., & Li, J. (2020). Guarantee the quality of service of control transactions in real-time database systems. *IEEE Access*, *8*, 110511–110522. https://doi.org/10.1109/ACCESS.2020.3002335

[19] Wu, P., & Ryu, M. (2017). Best speed fit EDF scheduling for performance asymmetric multiprocessors. *Mathematical Problems in Engineering*, *2017*(1), 1237438. https://doi.org/10.1155/2017/1237438

[20] TPC. (n.d.). *TPC download current specs/source*. Retrieved from: https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp

How to Cite: Noon, N. N., Getta, J., & Xia, T. (2024). Efficient Scheduling of Data Transfers in Multi-tiered Storage. *Journal of Data Science and Intelligent Systems*, *2*(4), 241–254. https://doi.org/10.47852/bonviewJDSIS42022471