OPTIMAL REMOTE OBJECT PLACEMENT IN DISAGGREGATED MEMORY SYSTEMS

by

Pranav Srisankar

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Electrical and Computer Engineering

Elmore Family School of Electrical and Computer Engineering

West Lafayette, Indiana

May 2026

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Vishal Shrivastav, Chair

Elmore Family School of Electrical and Computer Engineering

Dr. Xiaoqi Chen

Elmore Family School of Electrical and Computer Engineering

Dr. Mithuna Thottethodi

Elmore Family School of Electrical and Computer Engineering

Approved by:

Dr. Milind Kulkarni

ACKNOWLEDGMENTS

I would like to express my sincerest gratitude to my advisor, Professor Vishal Shrivastav, for his guidance, support, and insightful feedback throughout this work. I began my research journey with little prior exposure to the process, and his mentorship has been instrumental in shaping both my understanding of the field and my approach to research. His consistent encouragement and patience have played a significant role in my growth. I am especially grateful for his dedication and continued support, including his willingness to make time for discussions and provide guidance at all hours of the day (and night!). His commitment to mentoring has gone well beyond expectations, and his influence will continue to have a lasting impact on how I approach research and problem-solving as I continue into my PhD.

I am also thankful to Aneesh Kandi and Weigao Su for their collaboration and valuable discussions, which helped shape the ideas present in this work. They have been invaluable collaborators who supported me throughout my initial foray into research.

I would also like to thank my committee members for their guidance and support. Professor Mithuna Thottethodi has taught me a great deal about computer architecture through the courses I have taken, deepening my knowledge and intuition about how these systems operate. Professor Xiaoqi Chen has been an excellent professor to work with as a teaching assistant, providing me the freedom to grow significantly in my role. His mentorship has made teaching genuinely enjoyable and deepened my appreciation for it.

Finally, I am grateful to my family and friends for their continued support and encouragement throughout my studies. I could not have made it this far without them. To my closest friends, thank you for your friendship throughout this journey. The many moments we shared made my time at Purdue truly enjoyable, and I will always remember them.

Amma, Appa, and Nethra, thank you for your unwavering support and encouragement throughout this journey. I am deeply grateful for everything you have done for me. I would not be here without you.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Disaggregated systems decouple compute and memory, allowing compute nodes to access remote memory over bandwidth-constrained links. In such environments, each object consumes both storage capacity and network bandwidth, and placement decisions must be made online.

Object placement is formulated as an online two-dimensional bin-packing problem with a fixed number of nodes, aiming to maximize the number of accepted objects. Existing heuristics struggle to adapt to the online setting, often leading to premature resource exhaustion due to imbalanced resource utilization. To address these limitations, this thesis proposes DROP, a placement heuristic that jointly considers overall utilization and balance between storage and bandwidth. By guiding nodes toward balanced and unsaturated operating points, DROP preserves flexibility for future placements.

Through simulation across diverse workloads and system configurations, DROP was found to consistently improve object acceptance compared to standard baselines. Evaluations using synthetic traces show that DROP can improve object acceptance by up to 1.3×.

1. INTRODUCTION

Modern computing workloads have become increasingly memory-intensive, often requiring memory capacities that exceed what can be provisioned on a single machine. Disaggregated memory systems have emerged as a promising solution by decoupling compute and memory, allowing compute nodes to access remote memory pools over high-speed interconnects [1–5]. This architectural shift improves flexibility and resource utilization, but fundamentally changes how data placement must be performed.

In the disaggregated context, placing an object on a memory node consumes two distinct resources: storage capacity and network bandwidth. Storage determines whether the object can be held in memory at all, while bandwidth determines whether it can be served at the required rate. Unlike traditional systems, where capacity is the dominant constraint, disaggregated systems are also frequently limited by shared network bandwidth across many objects. As a result, placement selection must account for both dimensions simultaneously.

These decisions must also be made in an online setting. Objects arrive sequentially, and the system must decide immediately, if possible, where to place each object without knowledge of future arrivals. Once placed, objects are costly to move, making decisions effectively irreversible. Thus, the objective is to find feasible placements while preserving the system's ability to accommodate future objects.

We formulate this problem as a two-dimensional online bin-packing problem, where each memory node is a bin with fixed storage and network bandwidth capacities, and each object is characterized by its storage and bandwidth demands. Unlike classical bin-packing formulations that aim to minimize the number of bins used, our setting involves a fixed number of homogenous nodes and instead seeks to maximize the number of objects accepted over time.

To better understand the problem, we also introduce a geometric representation of the bin-packing problem, where the global system state is mapped in a normalized two-dimensional resource utilization space. In this interpretation, each node is represented by its storage and bandwidth utilization, and plotted as a point on the plane. To achieve perfect packing, nodes must move towards the top right corner of the unit square. This representation reveals that efficient placement depends not only on individual node utilizations but

also on maintaining a balanced and well-distributed set of nodes across the system. In particular, nodes that become saturated in one dimension while underutilized in the other lead to fragmented resources that cannot be used.

Existing heuristics are not well-suited to this objective. One-dimensional approaches such as First-Fit, Best-Fit, and Worst-Fit [6] consider only a single resource, often leading to imbalanced utilization where one resource is exhausted while the other remains underutilized. Two-dimensional heuristics improve on this by considering both resources jointly, typically through alignment-based metrics that match object demand to available capacity [7]. For example, schedulers such as Tetris use cosine similarity to align multi-resource demand vectors with available capacity [8]. However, these methods focus on local compatibility and do not explicitly account for how placement decisions shape the global distribution of resources across nodes. As a result, they often produce imbalanced states that reduce future placement flexibility, limiting long-term object acceptance.

Motivated by these observations, we propose **DROP** (**D**isaggregated **R**emote **O**bject **P**lacement), a placement algorithm that jointly optimizes resource utilization and balance. Rather than focusing solely on local compatibility or single-dimensional capacity, DROP evaluates the post-placement state of each candidate node and selects the one that best balances two objectives: remaining far from saturation and maintaining balanced utilization between both resource dimensions.

To evaluate this design, we build a simulation framework that models disaggregated memory systems with configurable storage and bandwidth constraints. We generate synthetic workloads with heterogeneous object sizes and bandwidth demands, spanning a wide range of operating regimes. Using this framework, we compare DROP against a variety of one-dimensional and two-dimensional heuristics.

Our results show that DROP enables a more balanced, concentrated distribution of node states, reducing resource fragmentation and preserving feasibility for future arrivals. We further demonstrate that this behavior enables DROP to outperform, or remain competitive against baselines across diverse workloads in terms of overall object acceptance.

2. MOTIVATION

In this chapter, we describe the motivation behind DROP.

2.1 System Model

This thesis considers a disaggregated memory system in which compute nodes access data hosted on homogeneous remote memory nodes over a high-speed interconnect. This model generalizes recent disaggregated architectures, in which memory is exposed as a shared resource pool rather than tightly coupled to compute.

The system stores and serves a collection of objects, where each object represents an arbitrary unit of data. Such objects arise naturally in modern systems, including key-value stores, caching layers, and data-intensive applications [9–11]. Unlike traditional memory systems, we do not assume a fixed granularity such as pages or cache lines. Instead, object sizes and access characteristics may vary widely depending on the workload.

Each memory node provides limited storage capacity and is connected to the network via a finite-bandwidth link. Serving requests for an object consumes both storage and network bandwidth. This abstraction is consistent with modern remote memory and disaggregated systems [1–4], which demonstrate that remote memory can be accessed efficiently at scale via RDMA-enabled interconnects.

2.2 Cost of Remote Data Access

Accessing remote data incurs overheads not present in local memory systems. Each request must traverse the network, incurring both latency and bandwidth costs.

While modern interconnects such as RDMA and CXL significantly reduce latency, network bandwidth remains a shared and limited resource [2, 12]. In practice, the gap between local memory bandwidth and network bandwidth is substantial. For example, a single CPU socket may sustain tens to one hundred GB/s of memory bandwidth, while a typical high-speed network link provides on the order of 100 Gbps ($\approx$ 12.5 GB/s) [13, 14]. Even with

multiple links or advanced fabrics, network bandwidth is still often an order of magnitude lower than aggregate memory bandwidth available within a memory node.

This disparity has important implications. A node may have sufficient storage capacity to hold additional objects, yet be unable to serve them at the required rate due to limited network bandwidth. Performance becomes network-bound under load [1, 2, 12, 15], even when memory capacity is not fully utilized.

Thus, each object demands two distinct resources:

- A storage cost, determined by its size
- A bandwidth cost, determined by its remote access demand

Both must be satisfied for the system to admit and serve the object without performance implications.

2.3 Related Works and their Limitations

Modern disaggregated memory systems and multi-resource schedulers provide important background for this work.

2.3.1 Multi-Dimensional Resource Constraints

The presence of multiple resource constraints fundamentally changes the placement problem. In disaggregated memory systems, each node is constrained by both storage capacity and network bandwidth. These constraints are physical limitations of the nodes.

This challenge has been extensively studied in the context of cluster scheduling, where jobs demand multiple resources such as compute, memory, and network bandwidth. Early work, such as Dominant Resource Fairness, formalizes fairness across multiple resource dimensions by equalizing each job's dominant share [16]. Production systems such as Google Borg [17], Apache Mesos [18], and Firmament [19] extend these ideas to large-scale datacenters, explicitly tracking multi-dimensional resource usage to improve utilization and fairness.

Beyond fairness, packing-based schedulers aim to improve utilization by matching job demands to available capacity. For example, the Tetris scheduler decides placements based

on the alignment between a job's demand vector and a machine's available resource vector [8]. This highlights the importance of placement decisions that consider the joint nature of resource consumption, rather than treating each dimension independently.

However, these systems operate in a different context than disaggregated memory placement. Cluster schedulers typically manage task placement with opportunities for rescheduling, preemption, or migration. Furthermore, they often operate in an offline setting, where pending jobs can be sorted based on demand [8]. In contrast, disaggregated memory placement decisions must be made online, complicating the placement problem. Furthermore, objects that are placed may exist in the system for a longer period. As a result, the goal shifts from fairness or instantaneous utilization to preserving long-term feasibility under streamed arrivals.

2.3.2 Disaggregated Memory Systems

Recent systems have demonstrated the feasibility and performance benefits of disaggregated memory architectures. These systems leverage high-speed networks such as RDMA or CXL to expose remote memory with low latency and high throughput, enabling applications to access memory beyond the availability in a single machine.

Infiniswap [1] is one of the earliest systems to demonstrate memory disaggregation using RDMA. It exposes remote memory as a swap device and distributes swapped pages across remote nodes using decentralized placement decisions, namely, power-of-two [20]. The primary focus of Infiniswap is scalability and load balancing across memory servers. However, placement is not formulated as a global optimization problem, and the system does not explicitly account for added resource constraints, such as network bandwidth, when determining where data should reside.

FastSwap [3] extends this idea by optimizing remote swapping performance through kernel-level mechanisms. It improves the latency and throughput of swap operations but also relies on OS-driven placement policies. These policies are designed for correctness and performance rather than for optimizing memory distribution across nodes under multiple constraints.

AIFM [2] takes a different approach by exposing far memory directly to applications and allowing them to control data placement and access. While this provides greater flexibility, the system focuses on application-level memory management and access efficiency. It does not provide a framework for coordinating placement decisions across nodes, nor does it explicitly account for multiple constraints, such as storage and bandwidth.

LegoOS [4] generalized disaggregation to a full operating system design, decoupling compute, memory, and storage into independent components. While LegoOS provides an abstraction for resource disaggregation, it does not define placement strategies that jointly optimize resource utilization across nodes. Memory is treated as a distributed resource pool, without regard for how placement decisions affect system-wide balance across multiple resource dimensions.

Beyond these representative systems, a wide range of disaggregated memory and resource disaggregation efforts exhibit similar characteristics. Systems such as HYDRA [21] and Canvas [15] focus on enabling scalable, high-performance access to remote resources and on addressing challenges in resource management, load balancing, and performance isolation. These works propose mechanisms to mitigate such issues, such as data movement, caching, or elasticity, but do not explicitly formulate object placement as an optimization problem. As a result, while these systems acknowledge and partially address the effects of resource exhaustion, they stop short of addressing the core question of how to optimally place objects to prevent these effects.

2.4 Implications

The combination of disaggregation, network-limited data access, and multi-resource limits creates a fundamentally different systems problem compared to traditional memory management. Efficient operation now depends on jointly managing storage and bandwidth to avoid imbalances across resource dimensions. These challenges motivate the need for placement strategies that explicitly consider both storage and bandwidth.

3. PROBLEM FORMULATION

We consider the problem of placing objects in a disaggregated memory system, where data is stored across a set of remote memory nodes and accessed over a shared network.

The goal is to maximize the number of objects that can be accepted under resource constraints. Each object demands both storage capacity and network bandwidth, and must be assigned to a memory node such that all resource requirements are satisfied.

This problem is inherently **online**. Objects arrive sequentially over time, and placement decisions must be made without knowledge of future arrivals. While the resource demand of each object is assumed to be known at arrival, the system has no information about future objects.

Once an object is placed, the decision is treated as irrevocable. This reflects the cost of data movement in disaggregated memory systems, where migrating data across nodes incurs additional latency and consumes network bandwidth. As a result, placement strategies must operate without relying on re-optimization.

3.1 Bin-Packing Formulation

We formalize the placement problem as a **two-dimensional bin-packing problem**.

Unlike classical bin-packing formulations, the number of bins (nodes, in this context) is fixed, and the objective is not to minimize the number of bins used, but to maximize the number of objects that can be accepted.

The challenge in this setting is not only satisfying constraints for individual objects, but doing so in a way that preserves the ability to accommodate future arrivals.

3.1.1 System Model

Let $M = \{1, 2, \ldots, m\}$ denote the set of available memory nodes in the system. Each node j $\in M$ is characterized by its available capacities:

- a storage capacity C_{j}, representing the maximum amount of data that can be stored on the node, and
- a bandwidth capacity B_{j}, representing the maximum network link bandwidth available for accessing data stored on that node, and which is shared across all objects placed on the node.

3.1.2 Object Model

Objects arrive sequentially over time, in an online fashion. Each object i is associated with:

- A storage demand $s_{\mathrm{i}} > 0$, and
- A bandwidth demand $b_{\mathrm{i}} > 0$, representing the cost and rate of remote accesses to the object.

This bandwidth demand must be met by the network links associated with the memory node on which the object is placed. Furthermore, we assume that the bandwidth demand of each object is known at the time of arrival.

3.1.3 Placement Model

Each object must be placed on exactly one memory node.

If object i is placed on node j, then:

- s_{i} units of storage are consumed by object i on node j.
- b_{i} units of network link bandwidth are consumed by object i on node j.

This requires placement decisions to simultaneously account for both storage and bandwidth consumption of the placed object.

3.1.4 Feasibility

A placement is feasible if, for every node:

1. the total storage assigned to that node does not exceed its storage capacity, and
2. the total bandwidth assigned to that node does not exceed its available link bandwidth.

More formally, let $x_{\mathrm{i,j}} \in \{0, 1\}$ be a binary variable such that $x_{\mathrm{i,j}} = 1$ if object i is placed on node j, and 0 otherwise.

A placement is feasible if the following constraints hold.

First, each object must be assigned to exactly one node:

$$\sum_{\mathrm{j} \in M} x_{\mathrm{i,j}} = 1 \quad \forall \mathrm{i} \tag{3.1}$$

Second, for every node, the total storage demand of assigned objects must not exceed its capacity:

$$\sum_{\mathrm{i}} s_{\mathrm{i}} x_{\mathrm{i,j}} \leq C_{\mathrm{j}} \quad \forall \mathrm{j} \in M \tag{3.2}$$

Third, for every node, the total bandwidth demand of assigned objects must not exceed the available link bandwidth:

$$\sum_{\mathrm{i}} b_{\mathrm{i}} x_{\mathrm{i,j}} \leq B_{\mathrm{j}} \quad \forall \mathrm{j} \in M \tag{3.3}$$

An object i is accepted if there exists at least one node $\mathrm{j} \in M$ for which assigning i does not violate the above constraints. Otherwise, the object is rejected.

These constraints enforce that a node must have sufficient available capacity in both dimensions for an object to be placed. Exhaustion in either dimension renders the node infeasible for future placements.

3.1.5 Problem Definition

We consider an online sequence of object arrivals over time. Upon the arrival of each object, the system must make an irrevocable decision to either accept and assign the object to a memory node or reject it.

Each object must be assigned to exactly one node if accepted. A placement is considered feasible if it satisfies the per-node storage and bandwidth capacity constraints defined in Equations 3.2 and 3.3. That is, for each node, the aggregate resource consumption of all assigned objects must not exceed the node's available storage and link bandwidth.

The objective is to maximize the total number of accepted objects over time.

Thus, the object placement problem is defined as a two-dimensional online bin-packing problem, where each object is a two-dimensional item defined by its storage and bandwidth demands, and each memory node is a bin with fixed capacities. Placement decisions must be made online and are irrevocable, with the objective of maximizing the number of accepted objects.

However, unlike classical formulations, the quality of a placement is determined not only by immediate feasibility but also by its impact on the system's ability to accommodate future arrivals.

3.2 Geometric Representation

The bin-packing formulation captures feasibility conditions and definitions but does not provide insight into how placement decisions affect long-term system behavior. In particular, it does not intuitively explain why certain placement strategies lead to higher acceptance rates than others.

To reason about these effects, we introduce a geometric representation of node utilization. This representation makes explicit how placement decisions influence both individual nodes and the global distribution of resources over time.

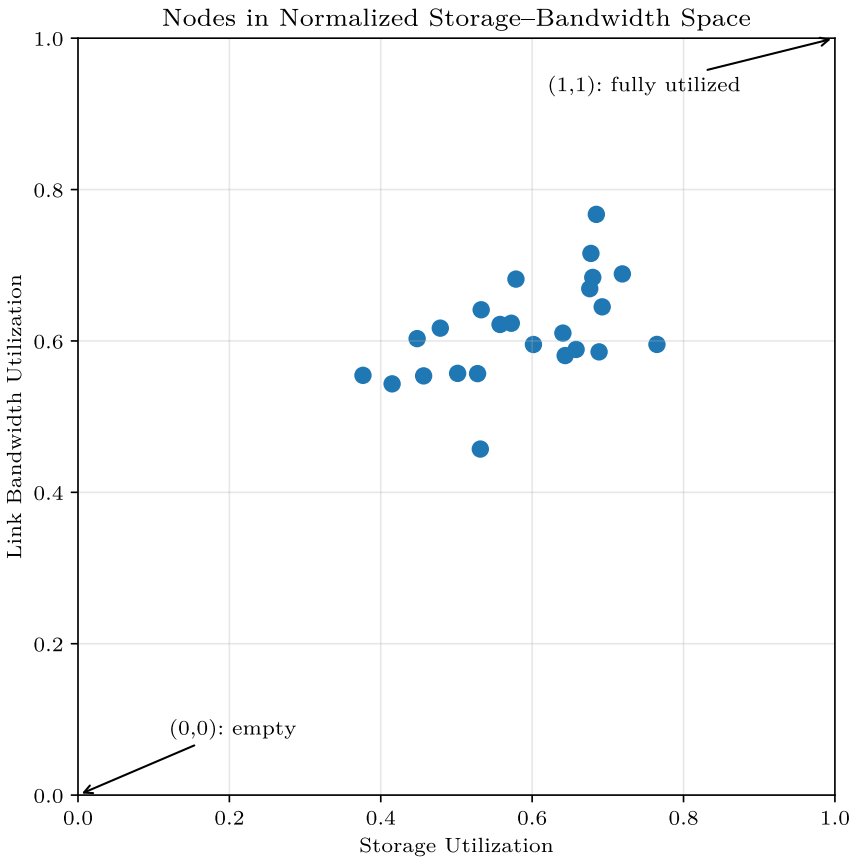

Figure 3.1. Geometric representation of memory nodes in the normalized storage and bandwidth space. Each point corresponds to a node, and the point (1, 1) represents full utilization of both resources.

3.2.1 Normalized Resource Space

We represent each memory node in a normalized two-dimensional space based on its current utilization.

For a node j, we define storage utilization as the storage used on node j divided by its total storage capacity, and bandwidth utilization as the bandwidth consumed on node j divided by its total bandwidth capacity.

Each node can therefore be represented as a point (x_j, y_j), where:

- $x_j \in [0, 1]$ represents storage utilization, and
- $y_j \in [0, 1]$ represents bandwidth utilization.

All nodes lie within the unit square $[0, 1] \times [0, 1]$.

This normalization treats storage and bandwidth symmetrically, enabling direct comparison between the two resource dimensions. As objects are placed, nodes move within this space, and the trajectory of these movements determines how system-wide resource utilization evolves over time. As shown in Figure 3.1, this space provides a compact representation of system-wide utilization.

3.2.2 Interpretation

This geometric space provides a direct visualization of system state, where:

- The origin $(0, 0)$ represents an empty node.
- The point $(1, 1)$ represents a fully utilized node.
- Points along either capacity boundary represent nodes where only one resource is fully utilized.

Placement decisions can therefore be viewed as a movement of nodes through this space. Different placement algorithms induce different movement patterns.

3.2.3 Feasibility Region for Incoming Objects

This geometric representation allows us to re-characterize placement feasibility.

Consider an incoming object with normalized requirements (s_i, b_i). A node at position (x_j, y_j) can accommodate the object if:

- $x_j + s_i \leq 1$, and
- $y_j + b_i \leq 1$.

Geometrically, this defines a feasible region consisting of nodes that are sufficiently far from any boundary. As nodes approach the edges of the square, the number of nodes in the feasible region shrinks, reducing the overall system's likelihood of accommodating future objects. Thus, we can maximize future acceptance by maintaining a larger set of feasible nodes. This relationship is visualized in Figure 3.2, where the feasibility region is shown relative to the object's requirements.

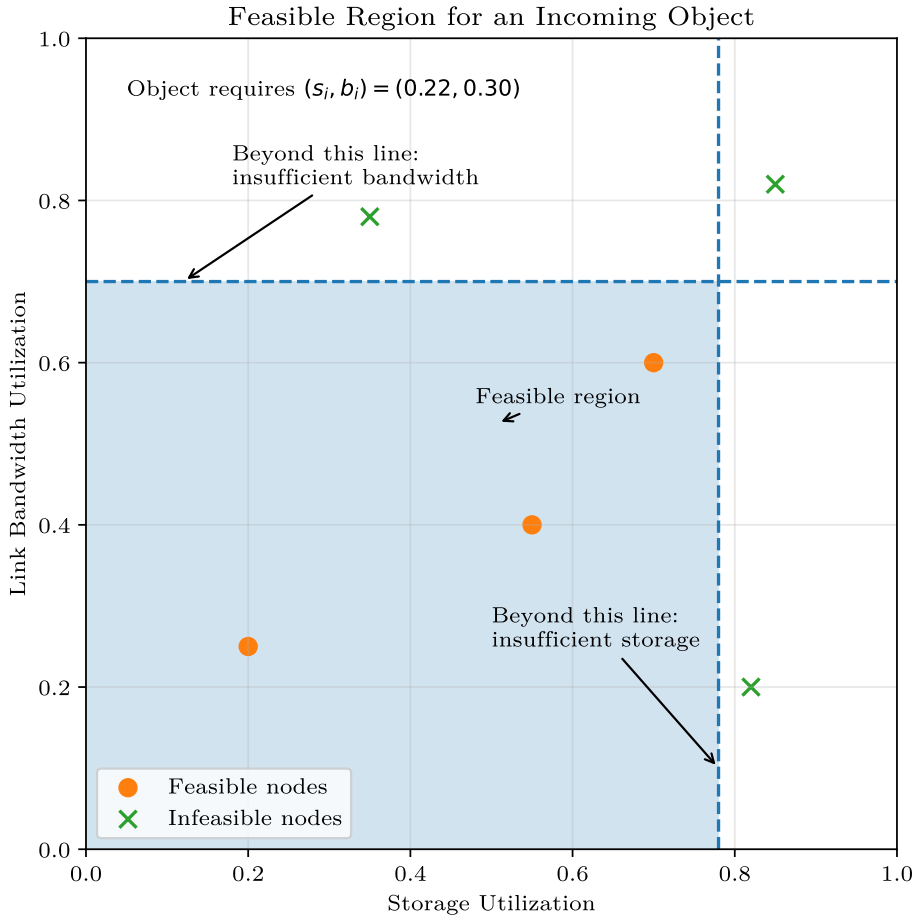

Figure 3.2. Feasible placement region for an object i with normalized requirements (s_i, b_i). Nodes outside the shaded region cannot accommodate the object.

3.2.4 Ideal Packing

The ideal node is one that approaches the point $(1, 1)$, indicating full utilization of both storage and bandwidth.

This corresponds to **perfect packing**, where no resources are wasted, and the system is maximally efficient.

However, in an online setting, maximizing utilization of individual nodes alone is not sufficient. System performance also depends on the distribution of nodes in the geometric space, not just their individual positions.

3.2.5 Resource Fragmentation

In reality, placement decisions can often lead to imbalanced utilization. Consider the following cases:

1. A node at $(1, 0.2)$: Storage is fully utilized, but link bandwidth remains largely unused.

2. A node at $(0.2, 1)$: Bandwidth is saturated, but storage remains underutilized.

In both cases, the remaining capacity in a dimension cannot be used because future objects require both storage and bandwidth simultaneously.

We refer to this phenomenon as **resource fragmentation**, in which available capacity for some resource is stranded and cannot be effectively utilized due to imbalances across dimensions.

This effect is illustrated in Figure 3.3, where nodes near the boundaries exhibit unused capacity in some dimension that is difficult to utilize.

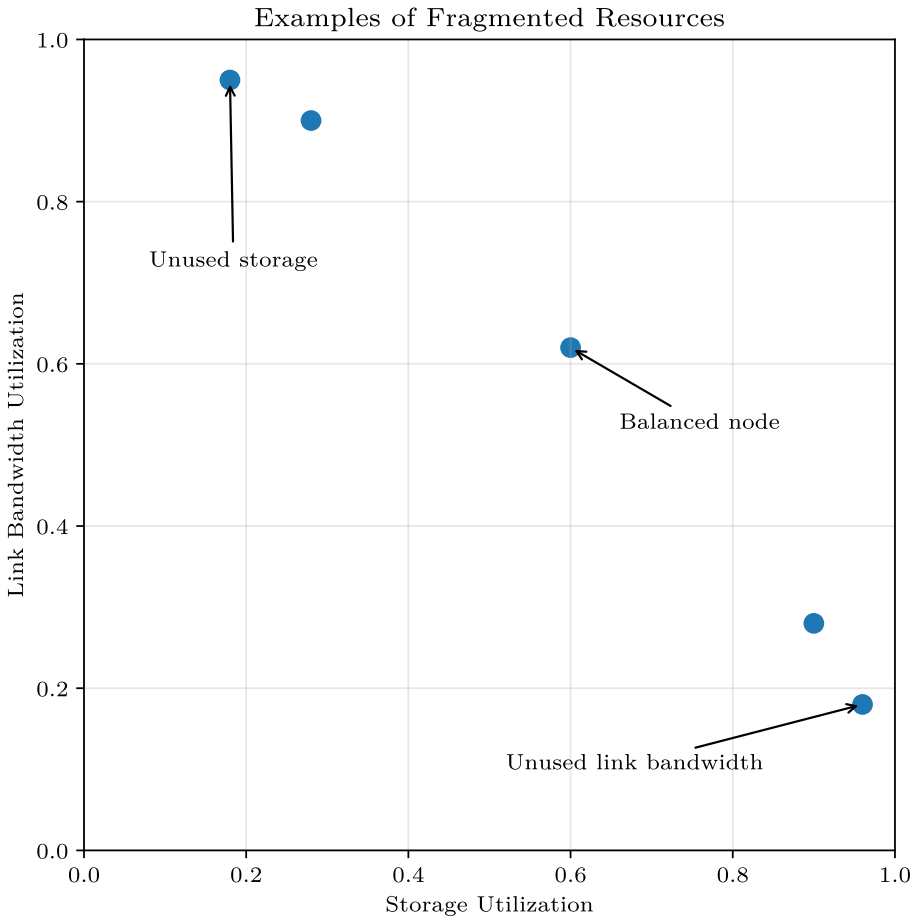

Figure 3.3. Examples of fragmented resources. Nodes near $(1, 0.2)$ and $(0.2, 1)$ have low capacity utilization in one dimension, which cannot be exploited because the other is saturated.

3.2.6 Distribution of Nodes and Fragmentation

Beyond individual nodes, system behavior is also determined by the distribution of nodes. Consider two global states:

- Clustered state: nodes are concentrated in a balanced region, such as (0.7, 0.7).
- Varied state: some nodes are near (1, 1), while others are near (0.2, 0.2) or along the axes.

Although a varied global state may contain some perfectly packed nodes, it can be less effective in an online setting.

This is because of three key reasons:

1. Fully utilized nodes cannot accept new objects, so the set of feasible nodes has decreased.

2. Underutilized nodes may lack sufficient capacity in one dimension.

3. Resources are unevenly distributed across nodes.

In contrast, a clustered configuration preserves balanced residual capacity across many nodes, increasing the likelihood that future objects can be placed.

This contrast is shown in Figure 3.4, where clustered nodes maintain consistent feasibility, whereas fragmented nodes result in unusable capacity.

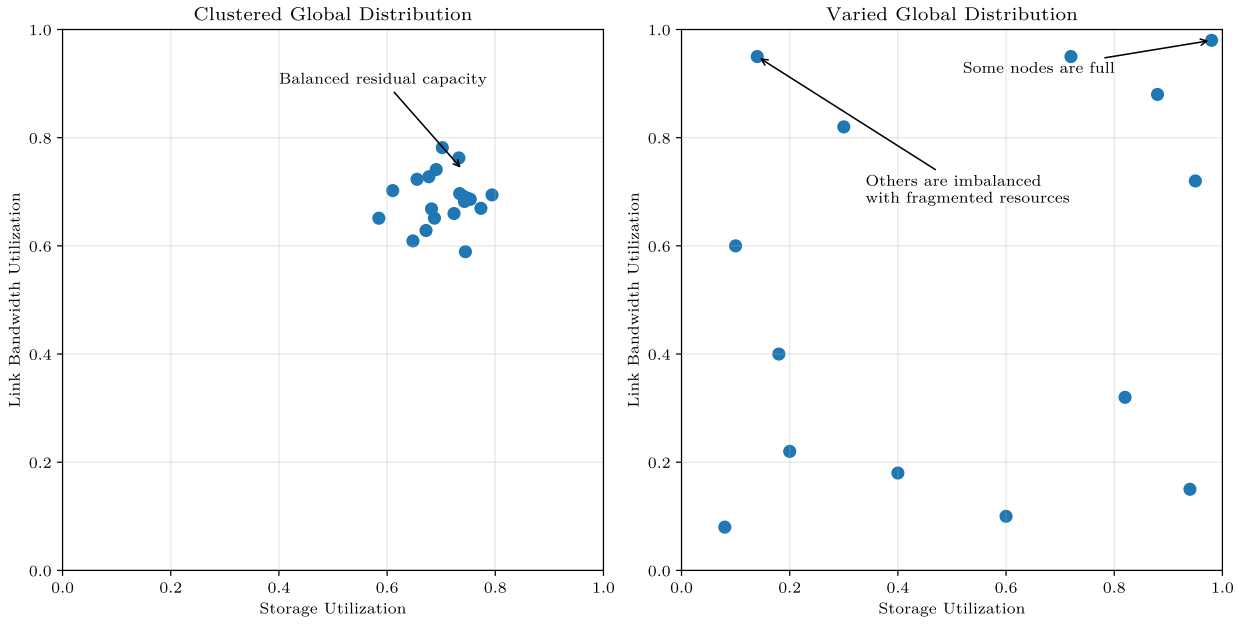

Figure 3.4. Clustered versus fragmented node distributions. A clustered distribution preserves balanced residual capacity across nodes, while a fragmented distribution leads to fragmented resources and reduced future placement flexibility.

3.3 Existing Bin-Packing Heuristics

We now discuss existing bin-packing heuristic algorithms applicable to the object placement problem. While these methods provide useful baselines, they are typically designed for different objectives than the one considered in this work. In particular, classical bin-packing focuses on minimizing the number of bins used, whereas our goal is to maximize the number of objects that can be accepted with a fixed number of nodes. This distinction significantly changes the behavior and effectiveness of different heuristics.

The underlying problem is computationally hard. The classical bin-packing problem is NP-hard, and our formulation extends it to a two-dimensional, online setting with additional constraints [22]. As a result, finding optimal placements is computationally prohibitive, especially when decisions must be made online. Accordingly, practical systems rely on efficient heuristic algorithms to make fast placement decisions.

3.3.1 One-Dimensional Heuristics

We first consider classical one-dimensional heuristics, which operate on a single resource dimension. In our setting, these heuristics can be applied to either the storage or the bandwidth.

The primary methods are First-Fit (FF), Best-Fit (BF), and Worst-Fit (WF) [6].

FF assigns each object to the first node that can accommodate it under some fixed ordering. While simple, it lacks any notion of placement quality. It does not attempt to balance utilization or preserve future capacity, and is highly sensitive to node ordering. As a result, it can lead to early imbalance across nodes.

BF assigns an object to the node with the smallest remaining capacity that can still accommodate it. In traditional bin packing, this strategy is effective because it tightly packs items, reducing the number of bins required. However, this objective does not align with our setting, where the number of nodes is fixed. Instead, the relevant objective is to preserve the ability to accommodate future arrivals.

BF also tends to create very small leftover capacities. While these fragments are technically unused capacity, they are often too small to be useful for future objects.

For example, consider nodes with remaining capacities $(0.9, 0.6, 0.4)$ and an incoming object of size 0.35. BF places the object in the node with 0.4 capacity, leaving $(0.9, 0.6, 0.05)$.

The remaining 0.05 is unlikely to be usable for future objects unless they are extremely small. Over time, BF produces many such fragments.

WF, in contrast, assigns an object to the node with the largest remaining capacity. In the same example, WF produces node capacities of $(0.55, 0.6, 0.4)$, preserving moderate residual capacity across all nodes.

Limitations of One-Dimensional Heuristics

Despite these differences, all one-dimensional heuristics share a fundamental limitation: they are only aware of one resource dimension.

In a two-dimensional setting, this leads to nodes that are saturated in one dimension but underutilized in the other. Geometrically, this corresponds to points near the line $x = 1$ or $y = 1$, resulting in resource fragmentation and reduced feasibility for future objects.

3.3.2 Two-Dimensional Heuristics

To address this limitation, several heuristics consider both storage and bandwidth simultaneously and are termed two-dimensional heuristics.

Two-dimensional heuristics consider both storage and bandwidth simultaneously when making placement decisions. Unlike one-dimensional methods, these heuristics evaluate candidate nodes using both resource dimensions, accounting for interactions between storage and bandwidth utilization.

In practice, many such heuristics aim to match an object's resource profile to that of a candidate node. Although they differ in their specific formulations, they share the common objective of selecting nodes whose available resources are most compatible with the object's demand across dimensions. We refer to this class of heuristics as alignment-based heuristics.

Let the demand vector of object i be $\vec{d} = \langle s_{\mathrm{i}}, b_{\mathrm{i}} \rangle$, and the available-resource vector of node j be $\vec{a_{\mathrm{j}}} = \langle SA_{\mathrm{j}}, BA_{\mathrm{j}} \rangle$, where SA_{j} and BA_{j} denote the available storage and bandwidth

on node j. Both vectors are normalized with respect to their corresponding node capacities, ensuring that all quantities lie within the range $[0, 1]$ and are comparable across dimensions.

Tetris (Cosine Similarity)

Tetris [8] measures alignment using cosine similarity between the demand vector and the available-resource vector:

$$\text{score(j)} = \frac{\vec{d} \cdot \vec{a}_{\text{j}}}{\|\vec{d}\| \cdot \|\vec{a}_{\text{j}}\|} \tag{3.4}$$

where the numerator is the dot product, and $\|\cdot\|$ denotes the Euclidean norm.

This formulation captures directional alignment, favoring nodes whose available resource vectors have similar directions to the object's demand vector.

In the canonical formulation of Tetris, the node with the highest cosine similarity is selected.

L_2-Norm-Ratio

The L_2-norm-ratio (L_2-Ratio) calculation compares demand relative to available capacity in each dimension [7]:

$$\text{score(j)} = (\frac{s_{\text{i}}}{SA_{\text{j}}})^2 + (\frac{b_{\text{i}}}{BA_{\text{j}}})^2 \tag{3.5}$$

Depending on the heuristic, nodes are selected by minimizing or maximizing this score. Canonically, the higher score is selected as that is a more perfect "fit", where the demand aligns more closely with available capacity.

L_2-Norm-Difference

The L_2-norm-difference (L_2-Diff) compares demand directly against available capacity using the difference [7]:

$$\text{score(j)} = (s_\text{i} - SA_\text{j})^2 + (b_\text{i} - BA_\text{j})^2 \tag{3.6}$$

Similar to L_2-Ratio, nodes can also be selected by minimizing or maximizing this score. Canonically, the lower score is selected as that is a more perfect "fit", where the demand aligns more closely with available capacity.

Limitations

The key limitation of alignment-based heuristics is that they optimize local compatibility between an object and a candidate node rather than the system's long-term global state.

These methods select nodes based on how well the object's demand vector matches the nodes' available-capacity vector, either in direction (cosine similarity), or in magnitude (L_2-based scores). While this captures feasibility and local fit, it does not explicitly reason about how a placement affects the system's ability to accommodate future objects.

Alignment-based heuristics implicitly treat progress toward saturation as beneficial, even when it occurs on already constrained nodes. However, these methods provide no mechanism to control how utilization moves nodes on the two-dimensional resource space over time, and therefore cannot ensure that capacity remains balanced or that future placement flexibility is preserved.

As a result, these heuristics may prematurely saturate individual nodes and produce states in which nodes drift toward boundary regions such as $(1, 0.2)$ or $(0.2, 1)$, where one resource is exhausted while the other remains underutilized. Once in these states, the remaining capacity becomes difficult to use, since future objects require both resources simultaneously.

For example, consider an object with normalized demand vector $\langle 0.2, 0.5 \rangle$, and two candidate nodes with available-capacity vectors $\langle 0.5, 0.55 \rangle$ and $\langle 0.9, 0.9 \rangle$. Under cosine-similarity

(Tetris) and both canonical L_2-norm scores, the node with the available-capacity vector $\langle 0.5, 0.55 \rangle$ would have been selected. However, placement into that node results in its available-capacity vector dropping to $\langle 0.3, 0.05 \rangle$. Now, bandwidth is effectively exhausted, and storage is fragmented.

This example highlights a limitation of the alignment-based heuristics: although they favor placements that appear to efficiently utilize resources for the current object, they do not control how nodes progress through the two-dimensional resource space. As a result, nodes may be pushed toward boundary regions where one resource is nearly exhausted, leaving the remaining capacity in the other dimension difficult to utilize. This leads to fragmented resources and reduces the system's ability to accommodate future objects.

3.4 Summary

In this chapter, we formulated the object placement problem in disaggregated memory systems as a two-dimensional online bin-packing problem. Each object consumes both storage and bandwidth, and must be assigned to a node without violating capacity constraints. Placement decisions are made online and are irrevocable, with the objective of maximizing the number of accepted objects.

We introduced a geometric representation in a normalized resource utilization space to reason about system behavior. This representation highlights that placement quality depends not only on individual node utilization, but on how utilization is distributed across nodes. In particular, imbalanced placements lead to fragmented resources, in which capacity in one dimension cannot be utilized due to saturation in another.

We examined existing bin-packing heuristics in this context. One-dimensional heuristics fail to account for interactions between storage and bandwidth, often resulting in imbalances across dimensions. Two-dimensional heuristics address this by considering both resources simultaneously, typically through alignment-based scoring functions. However, these methods focus on local matching between object demand and node capacity, and do not explicitly control the global distribution of resources across nodes.

As a result, existing heuristics do not directly address the underlying cause of inefficiency in this setting, namely, the fragmentation of resources across dimensions and across nodes.

This motivates the need for a placement strategy that explicitly accounts for system-wide resource balance. In the next chapter, we propose a new heuristic that takes these factors into account.

4. DESIGN

The placement strategy must assign each arriving object to a node such that the number of accepted objects is maximized over time.

From Subsection 3.1.4, an object can be placed on a node only if both storage and bandwidth constraints are satisfied. Each placement decision increases a node's storage and bandwidth utilization, thereby affecting whether future objects can be admitted.

Thus, the design objective is to select nodes such that both resource dimensions remain usable for future arrivals, without allowing one dimension to inhibit the other.

4.1 Design Principles

In this section, we discuss the design principles behind DROP.

4.1.1 Design Principle 1: Magnitude Principle

We begin with classical one-dimensional heuristics: First-Fit (FF), Best-Fit (BF), and Worst-Fit (WF). All three produce feasible placements when capacity is available. However, they differ in how they distribute remaining capacity, or utilization, across nodes.

FF is agnostic to current utilization except for feasibility decisions, while BF aims to maximize utilization in the chosen dimension.

WF assigns each object to the node with the lowest current utilization along the selected dimension. This avoids concentrating load into a small subset of nodes and instead preserves headroom across multiple nodes.

Why WF Can Outperform BF

The key distinction between BF and WF is not the tightness of packing, but the distribution of slack across nodes.

BF concentrates slack into very small fragments, while WF distributes slack more evenly across nodes. In other words, WF spreads the load, while BF concentrates it.

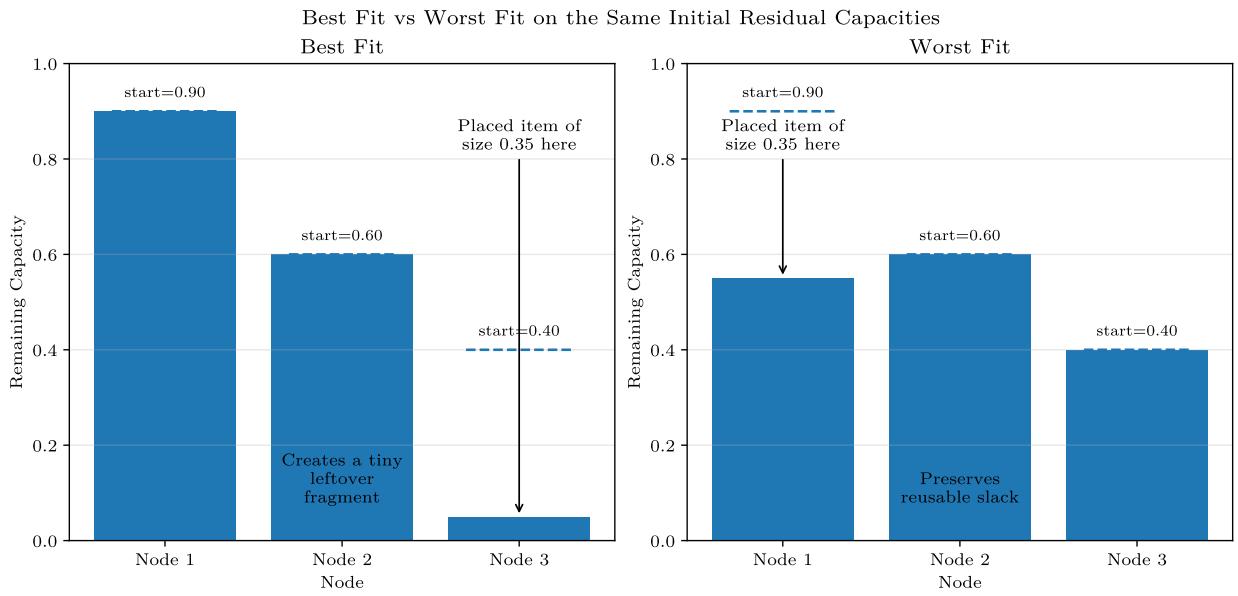

Figure 4.1. Best-Fit vs Worst-Fit in a one-dimensional setting. Over time, Best-Fit creates small, fragmented residual capacities, while Worst-Fit preserves larger, reusable slack across nodes.

In an online setting with unknown future arrivals, evenly distributed slack is more valuable than fragmented slack. Suppose future item sizes are random and not known in advance. A bin with remaining capacity r can accept a future item X with probability

$$P(X \leq r) \tag{4.1}$$

which increases with r. Thus, a system with several medium-sized residual capacities typically has greater future acceptance potential than one with many near-zero residual capacities.

This effect is illustrated in Figure 4.1, where BF produces unusable fragments while WF preserves usable capacity across nodes.

This provides a baseline principle:

Placement decisions should avoid driving nodes close to saturation prematurely.

Among the classical heuristics, WF directly enforces this behavior, making it a natural starting point.

One-Dimensional Worst-Fit

WF operates on a single resource dimension. In the two-dimensional setting, applying it to only storage or only bandwidth ignores the interaction between the two constraints.

A placement that preserves storage headroom may unknowingly exhaust bandwidth, or vice versa. As a result, nodes may remain far from saturation in one dimension but become unusable in the other.

Thus, while one-dimensional WF avoids early saturation in one dimension, it does not ensure that both resources remain usable after placement.

One-dimensional WF also induces a balancing effect across nodes. By always selecting the node with the largest remaining capacity, WF preferentially places objects on the least utilized nodes. Over time, this prevents load from concentrating on a small subset of nodes and instead distributes allocations more evenly. As a result, utilization levels across nodes tend to remain similar, implicitly reducing variance.

Naive Two-Dimensional Worst-Fit

To extend Worst-Fit to two dimensions, we first reduce the two resource utilizations to a single scalar measure. We then evaluate each node using its post-placement utilization.

In the geometric interpretation, let node j have a current utilization $(x_{\mathrm{j}}, y_{\mathrm{j}})$, and let object i have normalized demands $(s_{\mathrm{i}}, b_{\mathrm{i}})$. The post-placement utilization for each resource is:

$$x'_{\mathrm{j}} = x_{\mathrm{j}} + s_{\mathrm{i}} \text{ for storage, and}$$

$$y'_{\mathrm{j}} = y_{\mathrm{j}} + b_{\mathrm{i}} \text{ for bandwidth.}$$

A direct extension of WF is to select the node that minimizes overall utilization, or equivalently, maximizes remaining capacity. This can be implemented by selecting the node with the smallest magnitude of the utilization vector $\langle x'_{\mathrm{j}}, y'_{\mathrm{j}} \rangle$. We call this extension WF-2D.

This preserves the WF principle of keeping nodes away from saturation.

However, this extension considers only the overall utilization and does not account for skew in resource utilization.

To illustrate this limitation, consider three nodes with post-placement utilization vectors:

- $\langle 0.5, 0 \rangle$
- $\langle 0, 0.5 \rangle$
- $\langle 0.3536, 0.3536 \rangle$

All three have the same Euclidean magnitude of 0.5. As a result, they are indistinguishable under a magnitude-based rule.

However, they do not represent equally useful states. The first two nodes have highly imbalanced utilization, meaning future placements will be primarily constrained by a single resource dimension. In contrast, the third node preserves balance across both dimensions.

This demonstrates that magnitude alone does not capture whether a node remains well-positioned for future objects. The limitation arises because the magnitude-only placement is agnostic to the skew in resource utilization between the two dimensions.

This behavior is further illustrated in Figure 4.2, which shows a workload with 40% storage-heavy and 60% bandwidth-heavy objects. Under WF-2D, many nodes saturate in bandwidth while retaining unused storage capacity. As a result, nodes accumulate at the bandwidth boundaries where bandwidth is saturated, and storage remains underutilized. This leads to a wide spread in storage utilization across nodes, contradicting the balance effect of one-dimensional WF.

Thus, we must continue to build upon the naive WF-2D to have a more performant design.

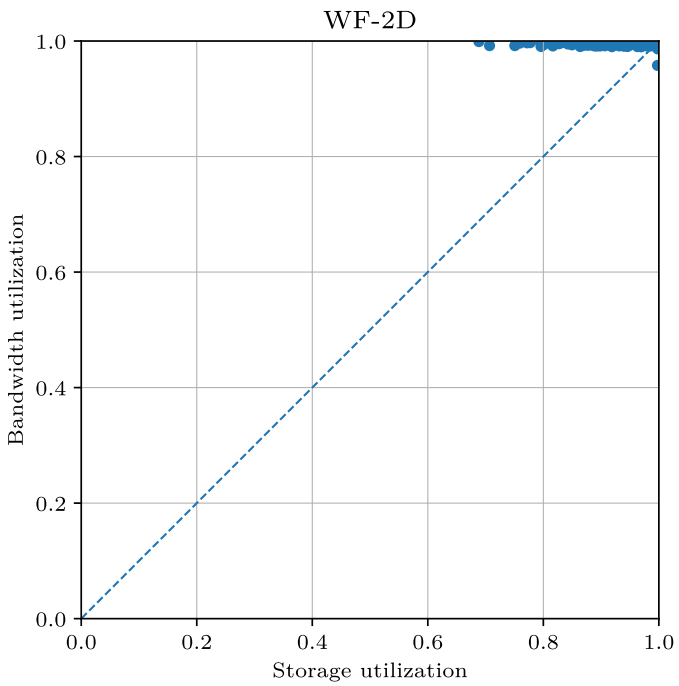

Figure 4.2. Node endpoint distribution under WF-2D showing skewed utilization and boundary accumulation.

4.1.2 Design Principle 2: Direction Principle

The previous examples show that nodes with similar overall utilization may differ in their ability to accept future objects. This difference arises from how utilization is balanced across dimensions.

Direction in the Geometric Interpretation

In the geometric representation, each node is a point (x_j, y_j) in the unit square. The ideal direction is toward $(1, 1)$, where both storage and bandwidth are fully utilized without inhibiting the other.

A placement decision updates the position of a node from (x_j, y_j) to (x'_j, y'_j).

If both dimensions increase to proportionally equal amounts, the node moves toward $(1, 1)$. If one dimension increases more than the other, the node moves toward a boundary.

Nodes that approach a boundary become constrained by one resource dimension before the other. Once this happens, the remaining capacity in the other dimension may no longer be usable for future objects.

This leads to the following additional principle:

Placement decisions must account for the direction of utilization, not only magnitude.

Direction-Aware Placement

To satisfy this requirement, we evaluate the post-placement utilization for each candidate node.

For a feasible node, the post-placement state (x'_j, y'_j) should also reflect a balance between the two dimensions' utilizations.

The balance condition corresponds to proximity to the line

$$x = y \tag{4.2}$$

Nodes near this line have similar storage and bandwidth utilization. Nodes far from this line are dominated by one dimension and are therefore more likely to become constrained early.

4.1.3 Summary

Together, we now have two design principles:

1. **Design Principle 1: Magnitude Principle.** The magnitude of a node's post-placement utilization must be minimized to preserve headroom for future object placements. In other words, the node should remain away from saturation.

2. **Design Principle 2: Direction Principle.** The direction of a node's post-placement utilization must be closely aligned to the $x = y$ line. In other words, the utilization of a node's two resources should remain balanced.

4.2 DROP Algorithm Design

We now describe DROP (*Disaggregated Remote Object Placement*), a placement algorithm that applies the design principles from the previous section. Together, these principles define the desired post-placement state of a node.

4.2.1 Post-Placement State

For each candidate node j, we evaluate the possible post-placement utilizations after placing object i:

- $x'_{\mathrm{j}} = x_{\mathrm{j}} + s_{\mathrm{i}}$
- $y'_{\mathrm{j}} = y_{\mathrm{j}} + b_{\mathrm{i}}$

Each node's post-placement state can therefore be represented as a point $(x'_{\mathrm{j}}, y'_{\mathrm{j}})$, where each dimension is normalized to the capacity of that resource.

Feasible nodes lie within the region:

- $x'_{\mathrm{j}} \leq 1$
- $y'_{\mathrm{j}} \leq 1$

A placement decision can thus be interpreted as moving node j from its current state $(x_{\mathrm{j}}, y_{\mathrm{j}})$ to its post-placement state $(x'_{\mathrm{j}}, y'_{\mathrm{j}})$ (Figure 4.3).

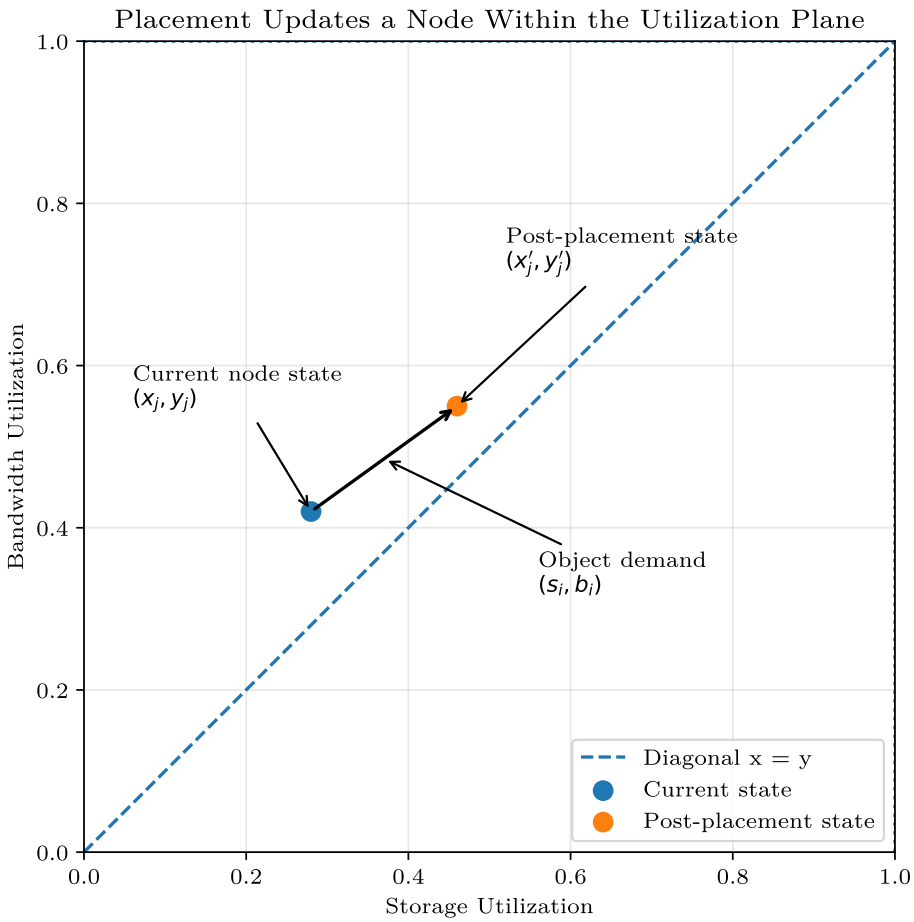

Figure 4.3. Placing an object i moves node j from its current state $(x_\mathrm{j}, y_\mathrm{j})$ to its post-placement state $(x'_\mathrm{j}, y'_\mathrm{j})$. The placement decision is therefore equivalent to selecting the resulting point after this movement.

4.2.2 Desired Geometry

The desired post-placement states satisfy two geometric properties:

- They lie away from the boundaries $x = 1$ and $y = 1$, indicating that the node remains below saturation.
- They lie close to the diagonal $x = y$, indicating balanced utilization between both resources.

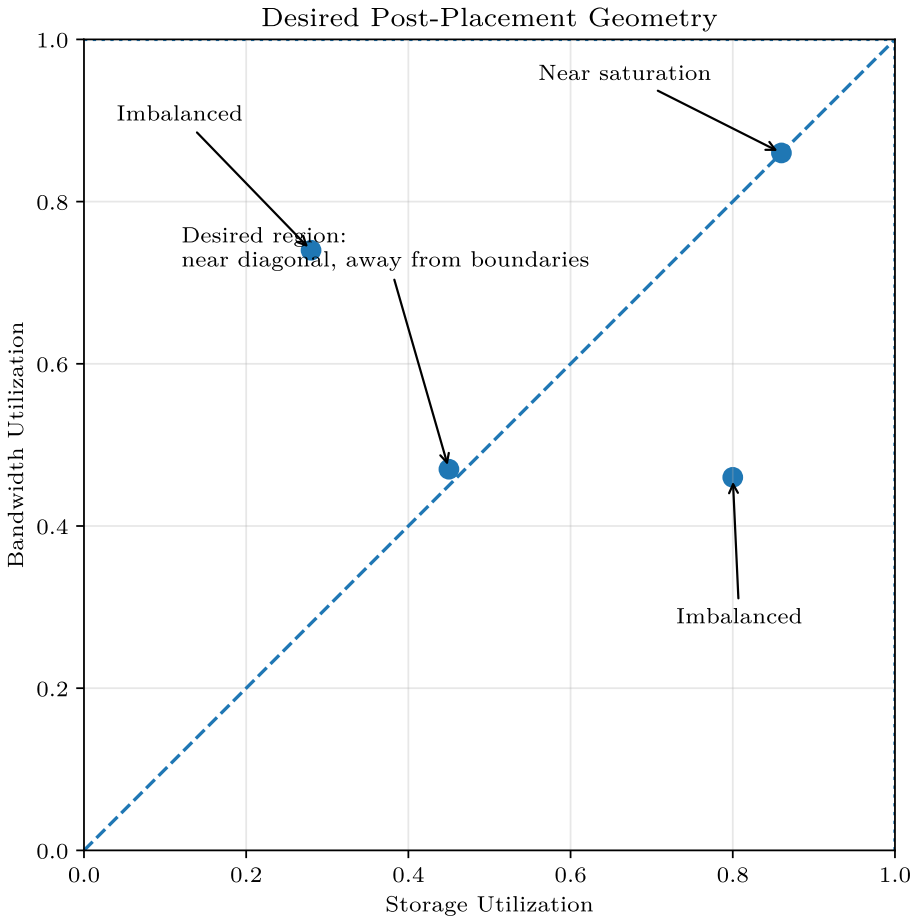

Figure 4.4. The desired region consists of points near the diagonal $x = y$ and away from the boundaries $x = 1$ and $y = 1$. Nodes in this region maintain both balanced utilization and remaining capacity.

The diagonal $x = y$ represents equal utilization between storage and bandwidth. Points far from this line are dominated by one resource dimension and are therefore more likely to become constrained early.

Thus, the placement objective can be viewed as selecting points that are both close to the diagonal and far from the boundaries, as shown in Figure 4.4.

4.2.3 Scoring Function

To implement this objective, we define a scoring function over candidate nodes based on their post-placement utilization.

The scoring function must capture the two quantities that represent the design principles:

1. Proximity to saturation.

2. Deviation from balanced utilization.

We now construct these components explicitly.

Components of the Score

We construct the score by evaluating the post-placement utilization (x'_j, y'_j) of each node.

Utilization Score (Magnitude Principle). The first component measures overall utilization:

$$\text{utilization score} = \sqrt{(x'_j)^2 + (y'_j)^2} \tag{4.3}$$

This corresponds to the Euclidean distance from the origin to the point (x'_j, y'_j), which in turn captures how close the node is to saturation. Here, smaller values indicate lower utilization and therefore more remaining capacity, consistent with the baseline principle.

Balance Score (Direction Principle). The second component measures deviation from balanced utilization:

$$\text{balance score} = \frac{|x'_j - y'_j|}{\sqrt{2}} \tag{4.4}$$

This corresponds to the perpendicular distance from the point (x'_j, y'_j) to the diagonal $x = y$, as illustrated in Figure 4.5. Here, a value of zero indicates perfect balance, while larger values indicate increasing imbalance between storage and bandwidth utilization.

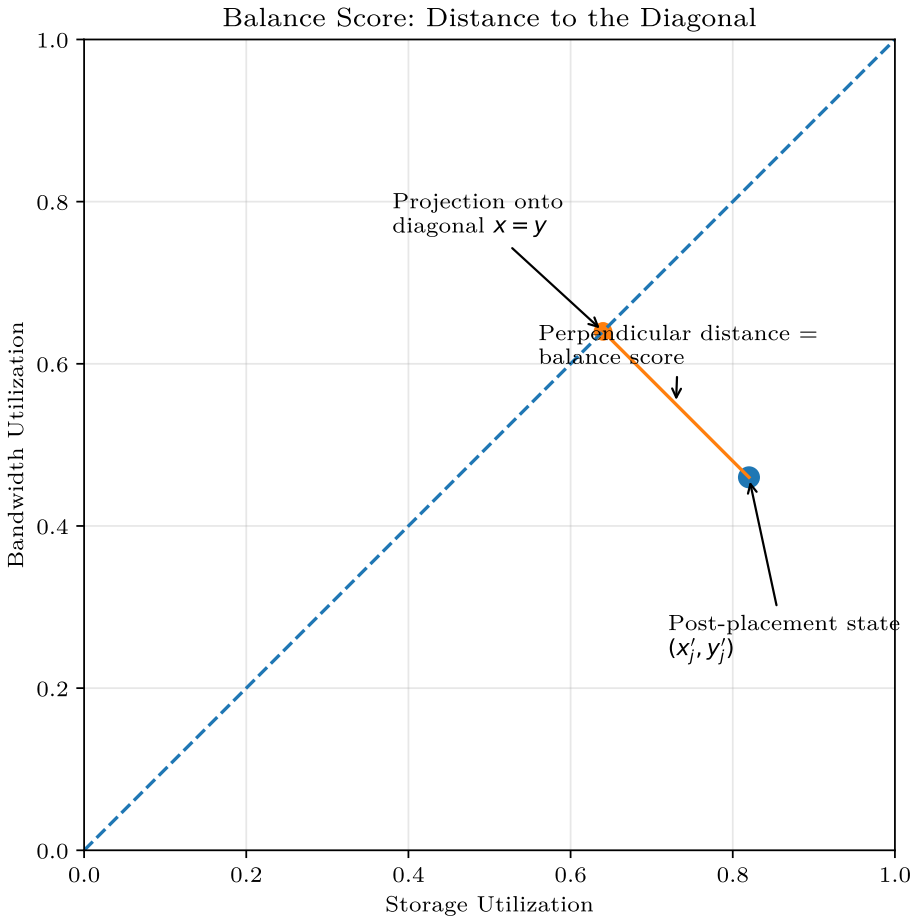

Figure 4.5. The balance score is the perpendicular distance from a node's post-placement state to the diagonal $x = y$, capturing imbalance between storage and bandwidth utilization.

Combined Score

These two components capture the complementary aspects of the placement objective. However, neither component alone is sufficient.

If we optimize only the utilization score, we would prefer nodes close to the origin, even when their utilization is highly imbalanced. Such nodes are likely to become constrained by a single resource dimension, violating the balance requirement.

If we optimize only the balance score, we would prefer nodes that lie exactly on the diagonal, even when they are significantly more utilized than other candidates. This violates the capacity preservation requirement.

The desired behavior is to select nodes that minimize the joint deviation from both objectives.

To illustrate this, consider three candidate nodes:

- A node whose post-placement state lies exactly on the diagonal but is relatively far from the origin.
- A node whose post-placement state lies slightly off the diagonal, but is much closer to the origin.
- A node whose post-placement state is close to the origin, but highly imbalanced.

The first node satisfies the balance condition but sacrifices capacity. The third node preserves capacity but is dominated by a single dimension. The second node provides the best compromise, remaining both relatively balanced and low in utilization.

This demonstrates that the placement decision should prefer nodes with small deviations in both dimensions, rather than nodes that optimize one objective while significantly violating the other, as shown in Figure 4.6.

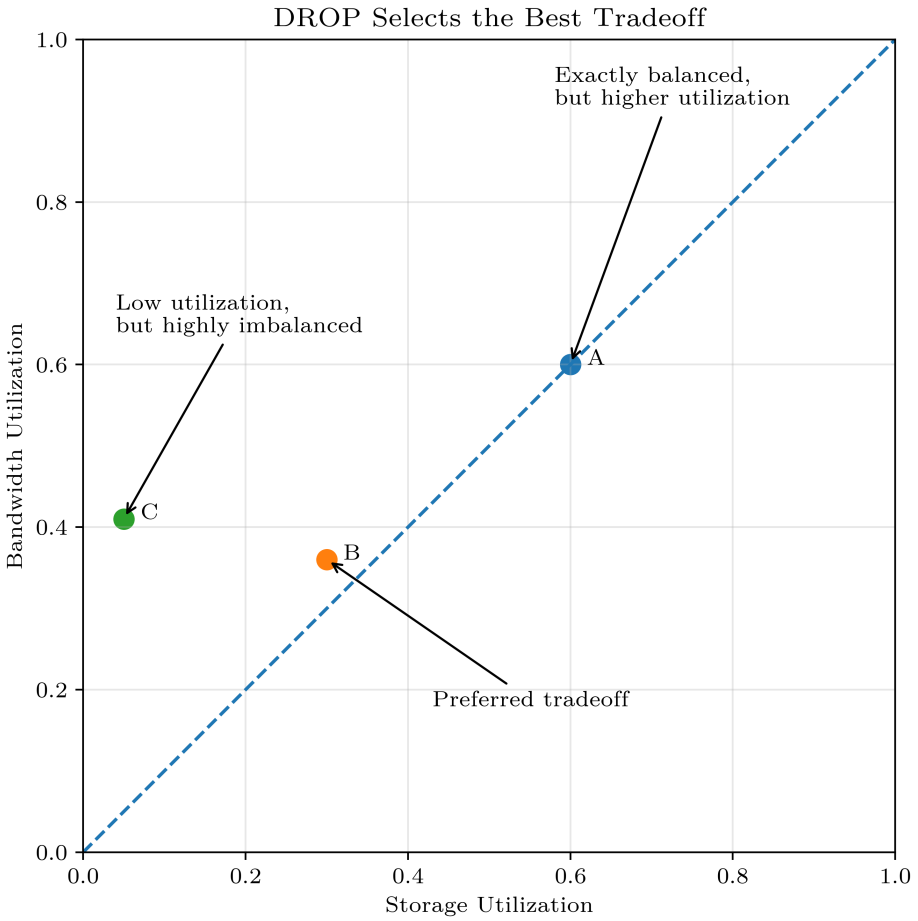

Figure 4.6. A node slightly off the diagonal but closer to the origin is preferred over nodes that optimize only one objective. DROP selects the node with the best joint tradeoff between balance and utilization.

This combination can be interpreted geometrically. Each node is mapped to a point in a transformed two-dimensional space, where one coordinate represents imbalance and the other represents overall utilization. In this space, the origin corresponds to the ideal state of zero imbalance and minimal utilization.

As shown in Figure 4.7, each node's post-placement state is converted into its corresponding balance and utilization coordinates.

The score of a node is then the Euclidean distance to the origin in this transformed space. This provides a symmetric and parameter-free way to compare candidates based on their joint deviation from both objectives.

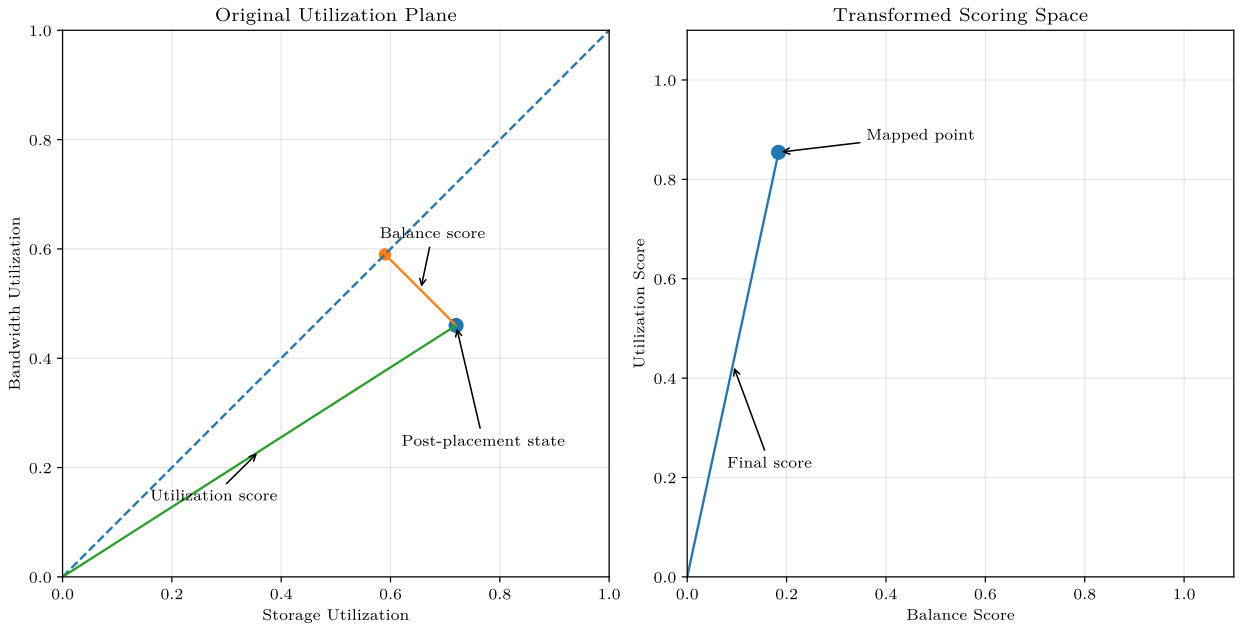

Figure 4.7. Each node is mapped from its post-placement state (x'_j, y'_j) to a transformed space where one axis represents imbalance and the other represents utilization. The final score is the distance to the origin in this space.

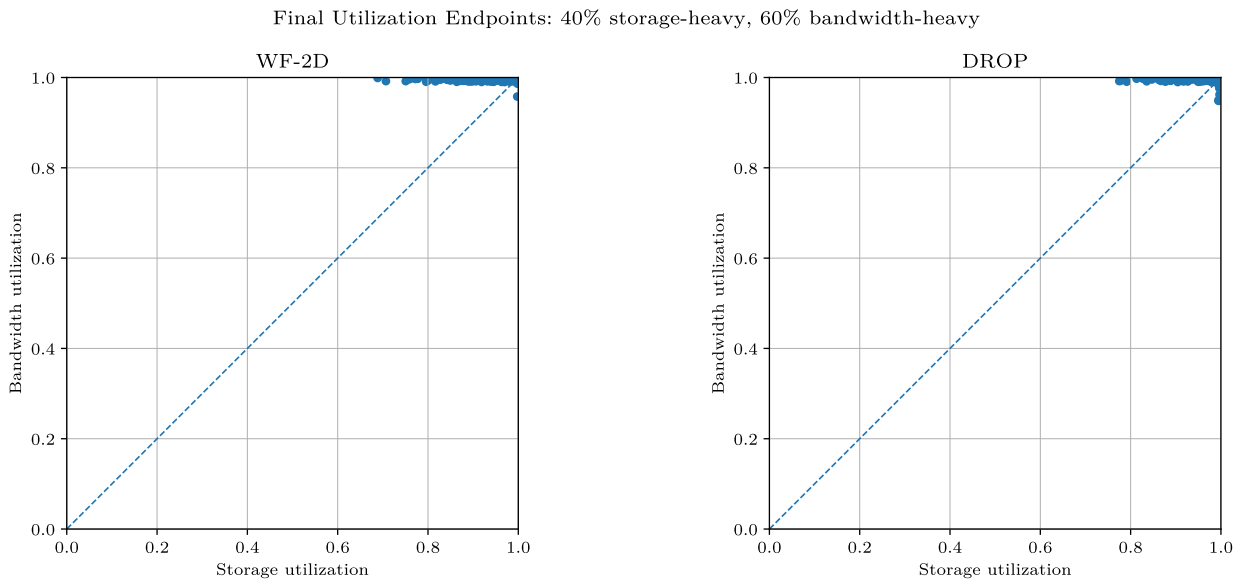

Figure 4.8. DROP maintains a concentrated distribution of node states, avoiding boundary accumulation observed in WF-2D.

This scoring rule has important implications at the system level. While DROP does not force every individual placement to lie exactly on the diagonal, it jointly penalizes both imbalance and high utilization. As a result, nodes that deviate significantly in either dimension become less likely to be selected.

Over repeated placements, this induces a concentration effect in the node-state distribution, similar to one-dimensional WF. As shown in Figure 4.8, DROP maintains node utilizations within a relatively narrow region of the resource utilization space, in contrast to WF-2D, which produces widely dispersed states that accumulate along boundary regions.

This concentration reduces fragmentation by avoiding nodes that are saturated in one dimension but underutilized in the other. Instead, capacity remains more uniformly distributed across both resources. Consequently, more nodes progress toward the balanced region near $(1, 1)$, while avoiding the high dispersion observed under WF-2D.

In other words, WF-2D preserves magnitude but not direction, whereas DROP preserves both.

This behavior is desirable because it preserves consistent feasibility across nodes. When node states are widely dispersed, individual nodes become constrained by different resources, leaving portions of capacity unusable. In contrast, a more concentrated distribution maintains balanced headroom across many nodes, increasing the likelihood that future objects can be placed.

Accordingly, we define the final score as:

$$\text{score(j)} = \sqrt{(\text{utilization score})^2 + (\text{balance score})^2} \tag{4.5}$$

For each arriving object, DROP evaluates all feasible nodes using this scoring function and selects the node with the minimum score. If no feasible node exists, the object is rejected.

5. EVALUATION

The goal of the evaluation is to determine whether DROP improves placement decisions in a two-dimensional setting, and whether these improvements translate into higher object acceptance.

We first validate that DROP produces more balanced and concentrated node utilization compared to a naive two-dimensional extension of Worst-Fit. We then evaluate overall algorithm performance by comparing object acceptance across a range of established placement heuristics.

5.1 Baselines

We compare DROP against a range of baseline heuristics that capture different approaches to online bin-packing in one and two dimensions.

5.1.1 One-Dimensional Heuristics

We first consider the classical one-dimensional heuristics described in Section 3.3, applied independently to each resource dimension:

- First-Fit (FF)
- Best-Fit (BF-Storage, BF-Bandwidth)
- Worst-Fit (WF-Storage, WF-Bandwidth)

These heuristics operate on a single dimension at a time and, therefore, ignore the interaction between storage and bandwidth utilization.

We also include a Random-From-Feasible-Subset (Random) baseline and a Round-Robin selection baseline. The Random baseline randomly assigns each object to all feasible nodes. The Round-Robin baseline assigns objects to nodes in a round-robin manner. These baselines separate the effect of placement strategy from feasibility, enabling analysis of whether structured placement decisions yield better long-term acceptance than random or ordered assignments.

5.1.2 Two-Dimensional Heuristics

We also consider heuristics that explicitly consider both dimensions when making placement decisions:

- Tetris (cosine similarity) (max, min)
- L_2-Ratio (max, min)
- L_2-Diff (max, min)

These methods attempt to align object demands with available node resources, typically by minimizing the distance or mismatch between demand and resource availability vectors. While they account for both dimensions, they do not explicitly target either low or balanced utilization. In particular, they do not directly enforce movement toward the diagonal while maintaining availability in both resources.

5.1.3 Relaxing Single-Node Placement

We also consider a problem formulation that relaxes the invariant from Subsection 3.1.3, allowing each object to be sharded and leading to multi-node placement. We present another baseline in which each object is equally sharded across all nodes, with its storage and bandwidth demands evenly distributed. While it leads to uniform utilization across all nodes and tight clustering, it does not necessarily mean that resource utilization is balanced in each node.

However, this baseline introduces new challenges in practical system implementations. Namely, this baseline requires coordination for multi-node placement and may incur overhead. Nonetheless, we include this baseline to provide insight into how idealized cross-node load distribution impacts utilization and object acceptance.

5.2 Experimental Methodology

All experiments are conducted in an online object-placement simulator in which objects arrive sequentially and must be placed immediately upon arrival. Placement decisions are made by the implemented baseline algorithm, and the state of all nodes (resource utilization, assigned objects) is tracked.

Each object arrives with a storage requirement and a bandwidth demand. A placement is considered feasible only if both constraints are satisfied on the selected node. If no feasible node exists, the object is rejected.

To approximate online behavior, each experiment uses a sufficiently long sequence of object arrivals such that the system always reaches saturation by the end of the simulation. Algorithm performance is measured by the total number of objects successfully placed before no feasible placements remain. This metric reflects the system's effective capacity under each placement algorithm.

5.2.1 Workload Model

Workloads are constructed using three fundamental object types:

- Storage-heavy: objects whose storage demand dominates bandwidth demand
- Bandwidth-heavy: objects whose bandwidth demand dominates storage demand
- Balanced: objects with proportional storage and bandwidth demand

Using these classes, we generate several workload distributions by varying their relative proportions. We sweep the percentage of storage-heavy objects from 0% to 100%. For each such configuration, we also sweep the percentage of bandwidth-heavy objects. The remaining objects are assigned to the balanced class. This parameterization spans the full range of workload compositions, from purely storage-heavy to purely bandwidth-heavy regimes, as well as mixed workloads with varying degrees of imbalance.

In addition to workload composition, object sizes and bandwidth demands are drawn from two distributions: a uniform distribution and a heavy-tailed Pareto distribution. The

uniform distribution generates objects with evenly distributed sizes and bandwidths within a fixed range, whereas the Pareto distribution introduces skew, producing few large, high-demand objects alongside mostly smaller ones. These distributions capture both balanced and heavy-tailed workload characteristics commonly observed in practice. Furthermore, we run five independent trials for each workload configuration, each with a different random seed.

All workloads use a fully randomized arrival order. Objects are shuffled prior to simulation, and all algorithms are evaluated on identical traces with identical arrival order.

5.2.2 Evaluation Metrics

Before evaluating the algorithmic performance of all baselines, we first validate that DROP improves overall per-node balance and variance.

To quantify this effect, we analyze the distribution of node utilizations produced by DROP relative to each baseline algorithm. For each node, we compute the harmonic mean of its normalized storage and bandwidth utilization, which captures overall utilization while penalizing imbalance between the two resources.

We then examine the variance of node utilizations to measure their dispersion. Lower variance indicates that nodes are concentrated in a narrower region of the utilization space, while higher variance indicates that nodes are spread out.

Finally, we measure the algorithmic performance of DROP relative to each baseline algorithm. The primary metric is the total number of objects successfully accepted over the simulation's lifetime. Since all experiments run until no feasible placements remain, this directly captures the effective utilization of system resources.

5.3 Experimental Setup

All experiments are conducted using identical system configurations and workloads. We set the following system parameters:

- Number of memory nodes: 128
- Storage capacity per node: 128 GiB
- Bandwidth capacity per node: 100 Gbps

5.4 Resource Utilization Distribution

We first analyze how different placement heuristics shape the distribution of node resource utilization across the system. While total object acceptance captures how effectively a policy uses system capacity, it does not explain how that capacity is distributed across nodes. This analysis provides insight into the structure of node states produced by each policy.

Since all measurements are taken after the system reaches saturation, nodes can no longer accept additional objects. As a result, low harmonic means primarily reflect imbalance between resources, rather than simply low overall utilization. Such nodes correspond to states where one resource is exhausted while the other remains underutilized.

We then measure the variance of these values across nodes to quantify how dispersed node states are under each baseline algorithm. Lower variance indicates that nodes are concentrated in a narrower region of the resource space, while higher variance indicates more widely spread nodes.

For ease of comparison across algorithms, we normalize both metrics in the plots. Mean utilization is normalized such that higher values indicate more balanced utilization, while lower values of normalized variance indicate better performance.

Figures 5.1-5.15 show the normalized mean and variance of per-node utilization balance for DROP relative to each baseline. Across all baselines, several consistent trends emerge.

First, DROP consistently achieves lower variance while maintaining comparable or higher mean balance. This indicates that it not only utilizes capacity effectively, but also keeps node states more concentrated in the two-dimensional resource utilization space.

Second, the variance gap is the largest compared to one-dimensional heuristics. These algorithms optimize only a single resource, and while they may reduce variance in one dimension, they effectively increase variance in the other. This results in highly dispersed node states and fragmented capacity.

Third, Worst-Fit variants reduce this dispersion relative to First-Fit and Best-Fit by spreading load across nodes, but they still produce skewed node states.

Finally, when compared to two-dimensional alignment-based heuristics such as Tetris and L_2-based heuristics, DROP continues to exhibit lower variance while maintaining strong mean utilization. Although these methods consider both dimensions, they optimize the local compatibility between object demand and node availability, rather than controlling the global evolution of node states. As a result, node states remain more dispersed and less consistently balanced.

Overall, these results show that DROP maintains a more balanced and concentrated node-state distribution across workloads. This concentration preserves usable capacity in both resource dimensions and helps explain the higher object acceptance observed in subsequent results.

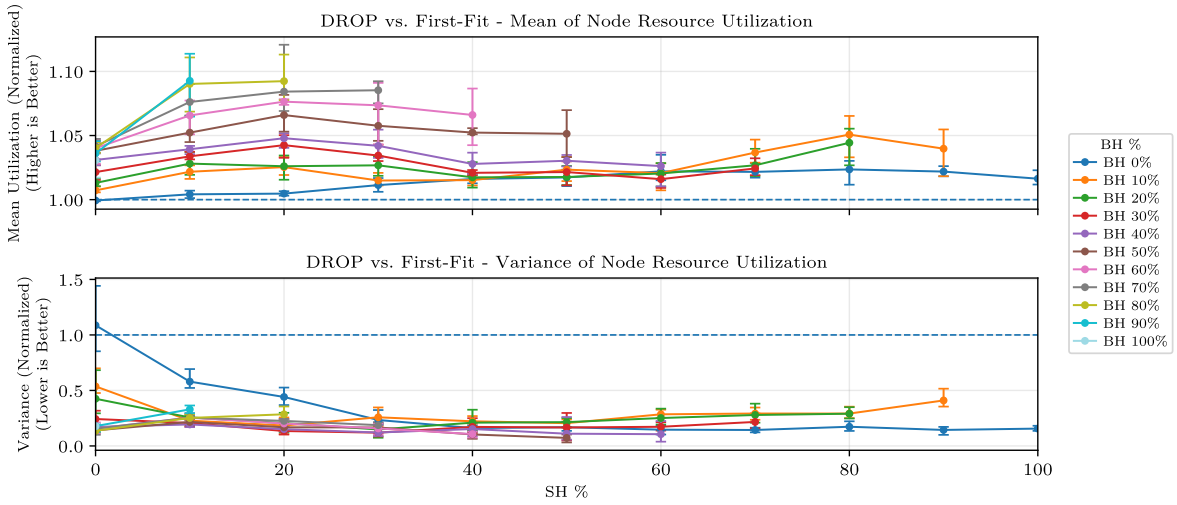

Figure 5.1. Normalized mean and variance of per-node utilization for DROP relative to FF.

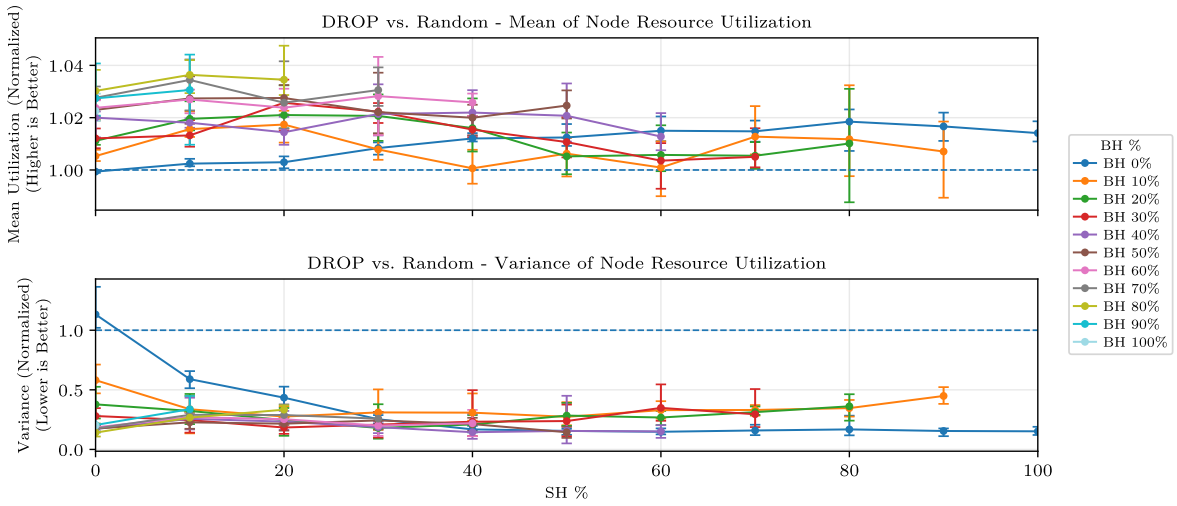

Figure 5.2. Normalized mean and variance of per-node utilization for DROP relative to Random assignment.

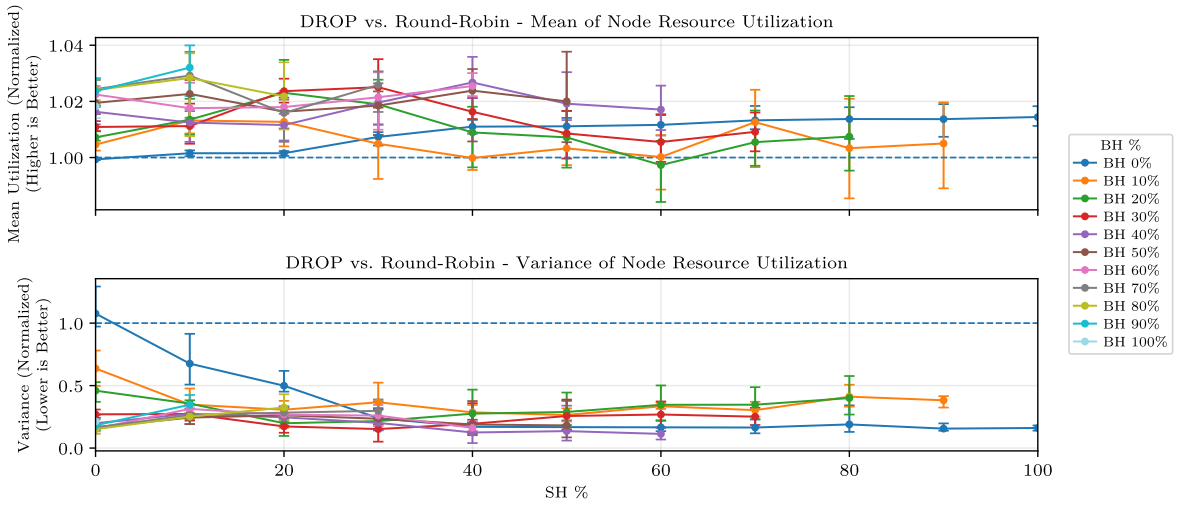

Figure 5.3. Normalized mean and variance of per-node utilization for DROP relative to Round-Robin assignment.

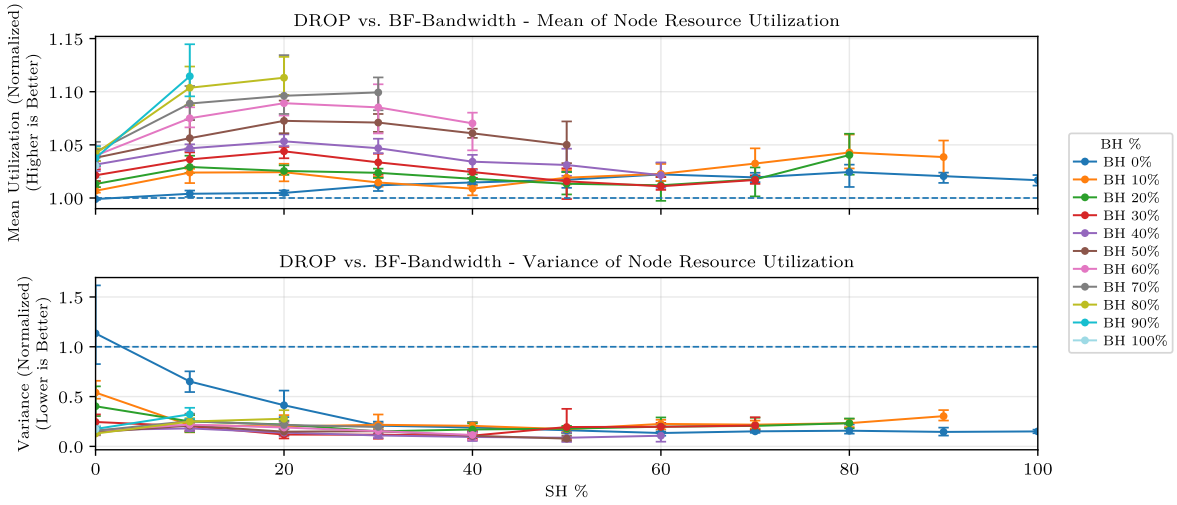

Figure 5.4. Normalized mean and variance of per-node utilization for DROP relative to BF-Bandwidth.

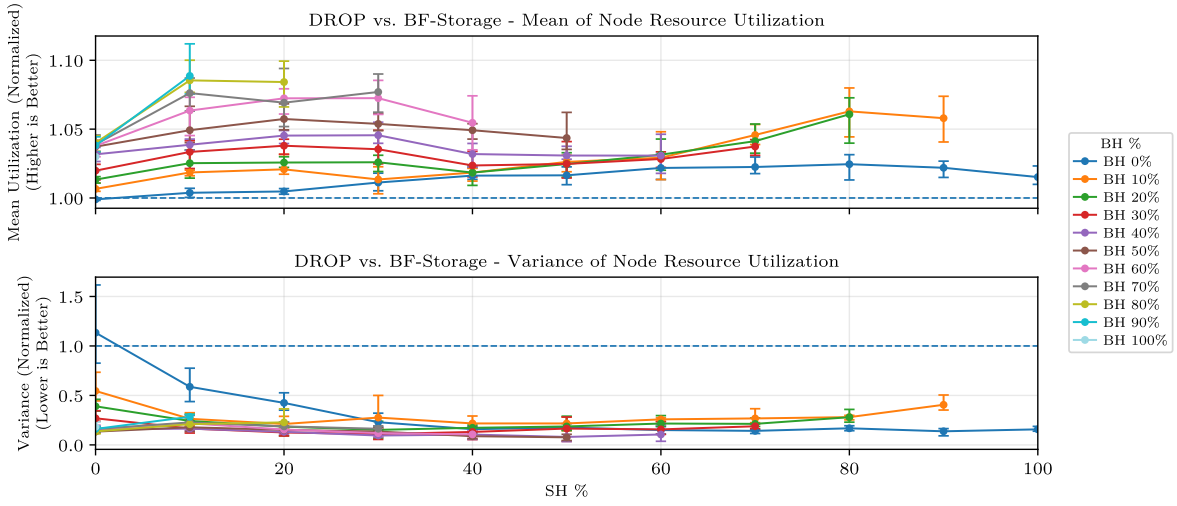

Figure 5.5. Normalized mean and variance of per-node utilization for DROP relative to BF-Storage.

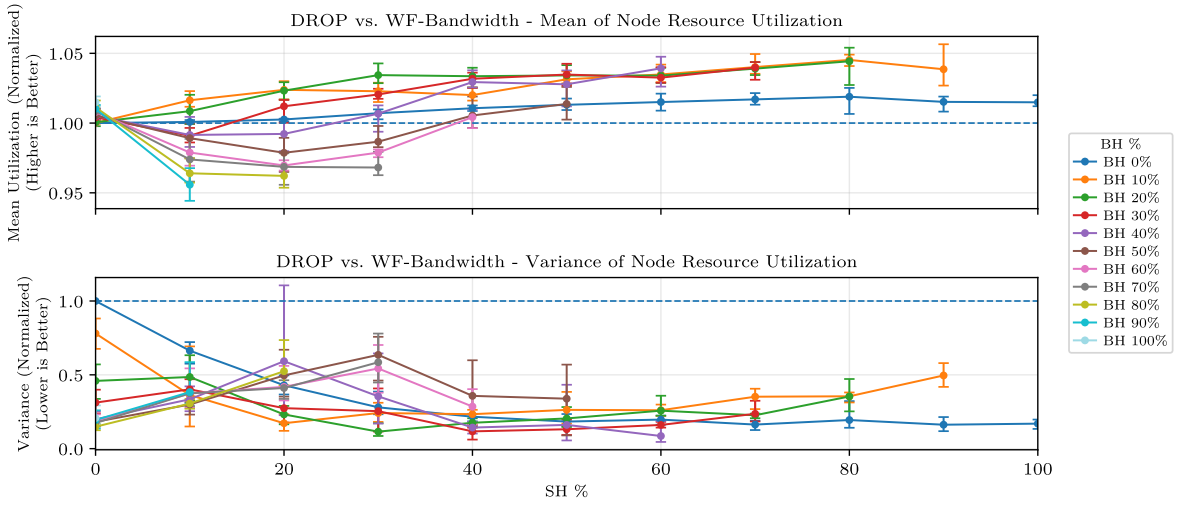

Figure 5.6. Normalized mean and variance of per-node utilization for DROP relative to WF-Bandwidth.

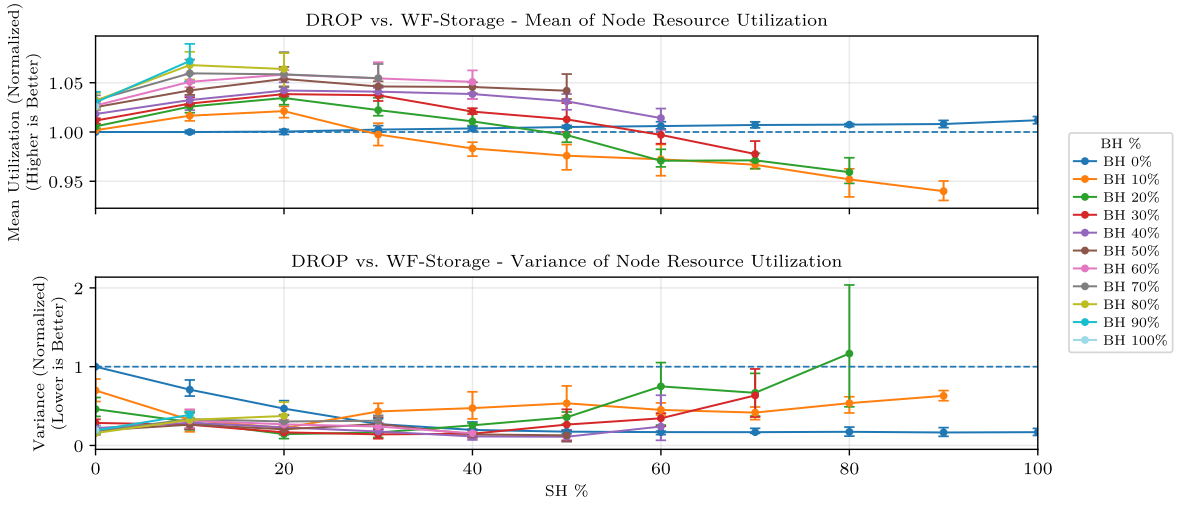

Figure 5.7. Normalized mean and variance of per-node utilization for DROP relative to WF-Storage.

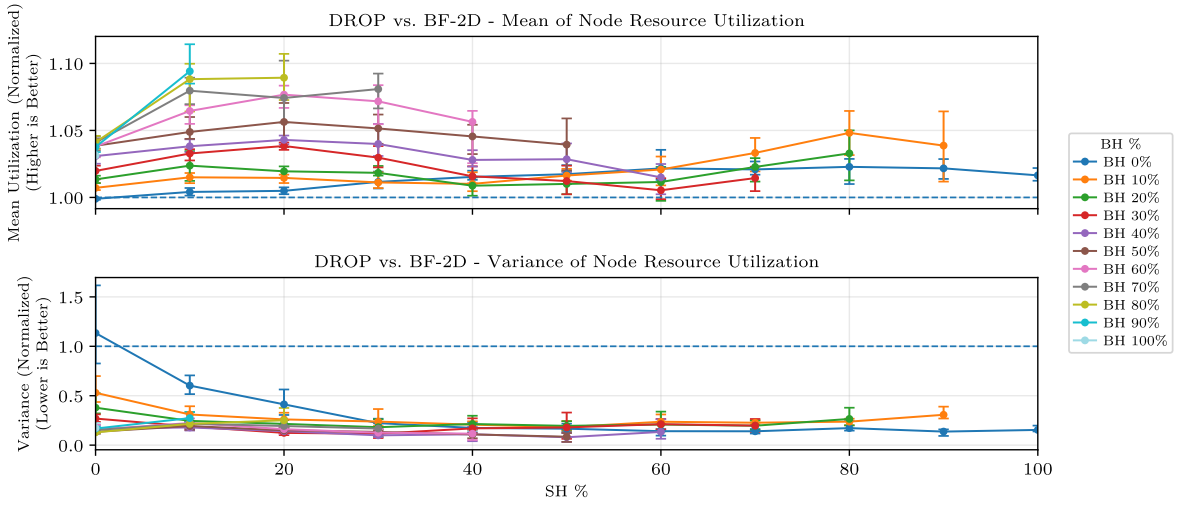

Figure 5.8. Normalized mean and variance of per-node utilization for DROP relative to BF-2D.

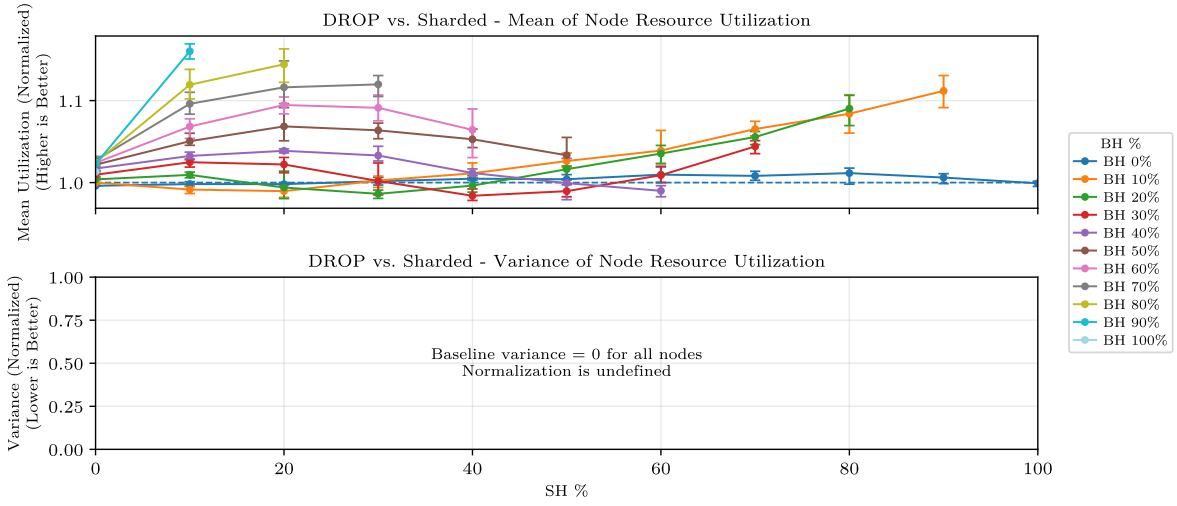

Figure 5.9. Normalized mean and variance of per-node utilization for DROP relative to naive sharding.

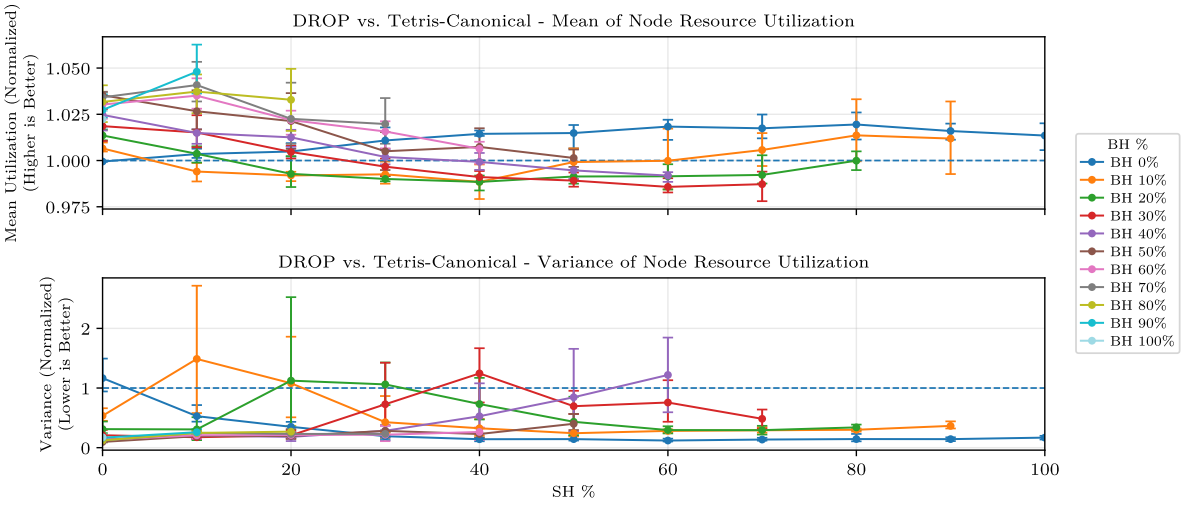

Figure 5.10. Normalized mean and variance of per-node utilization for DROP relative to Tetris-Canonical.

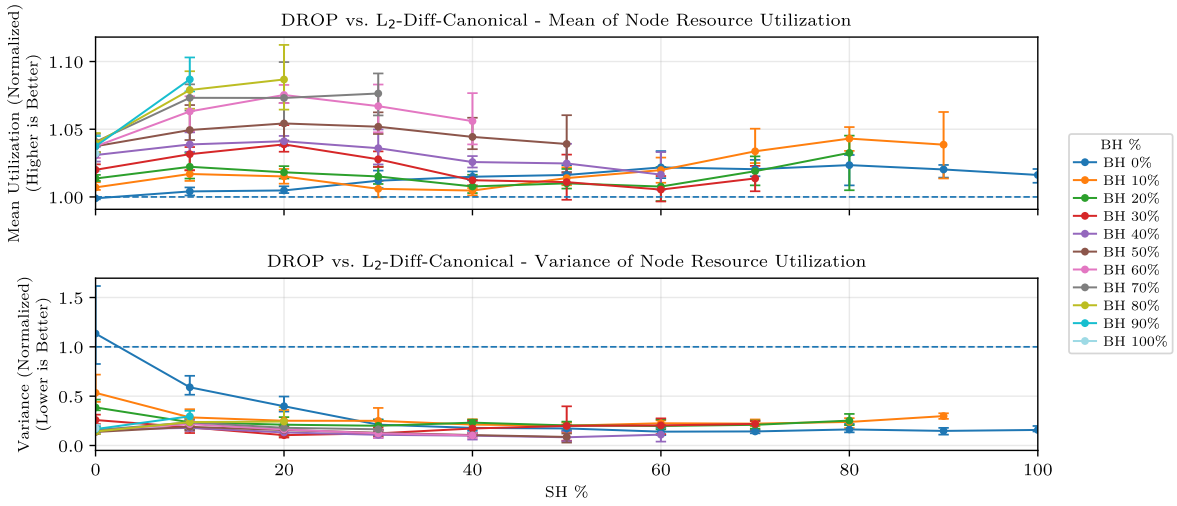

Figure 5.11. Normalized mean and variance of per-node utilization for DROP relative to L_2-Diff-Canonical.

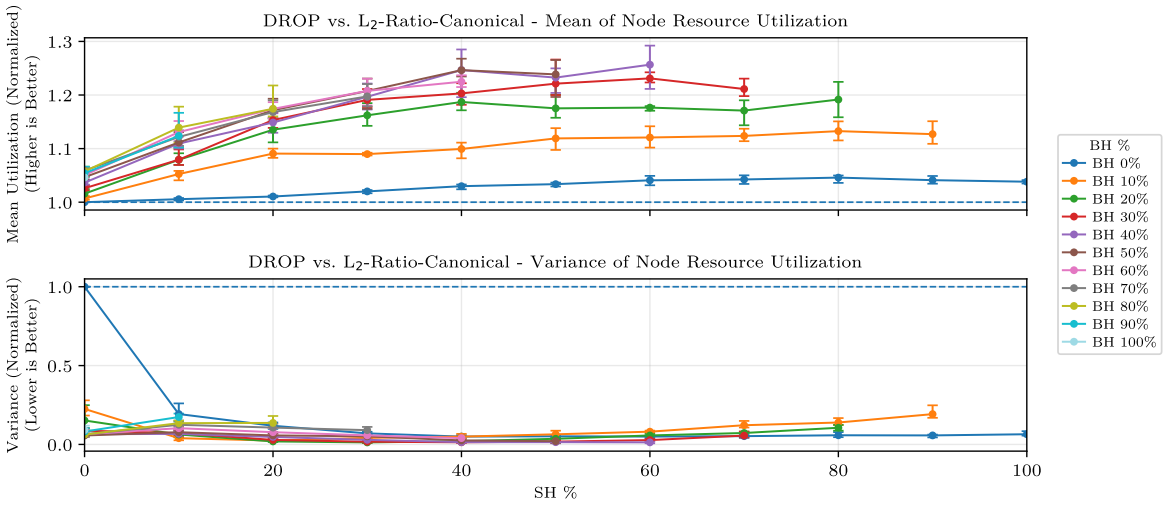

Figure 5.12. Normalized mean and variance of per-node utilization for DROP relative to L_2-Ratio-Canonical.

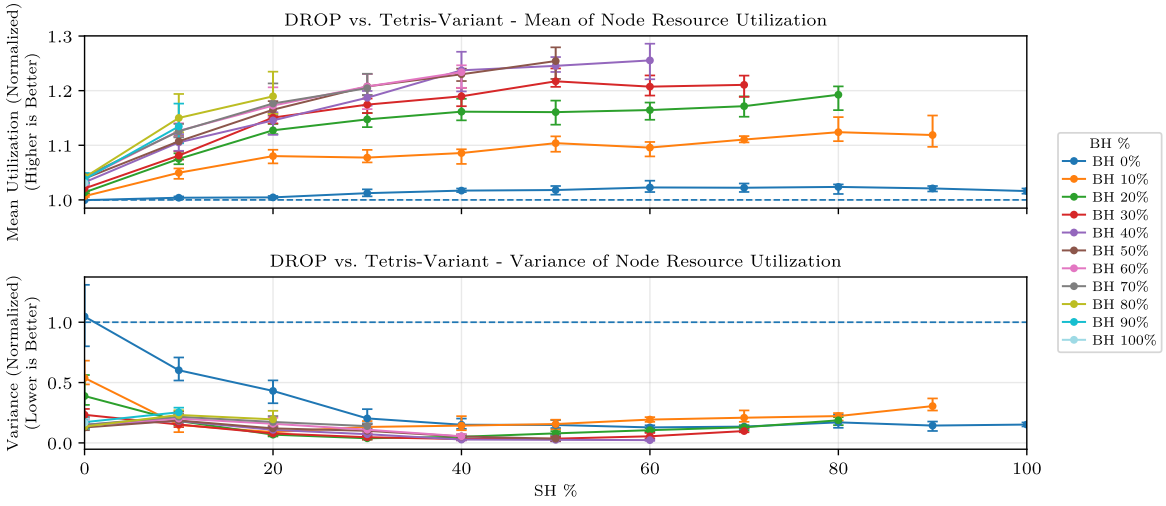

Figure 5.13. Normalized mean and variance of per-node utilization for DROP relative to Tetris-Variant (min).

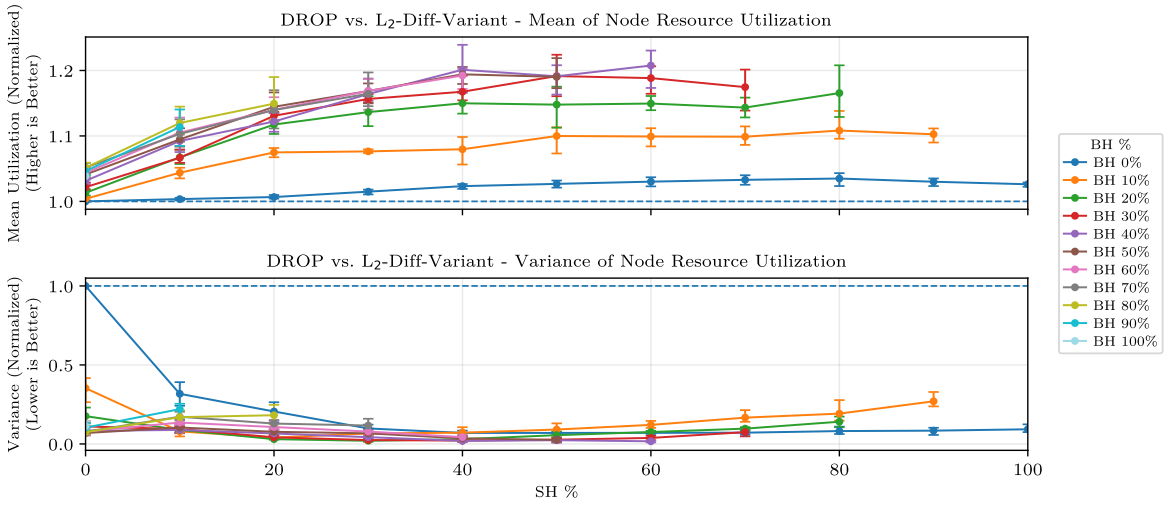

Figure 5.14. Normalized mean and variance of per-node utilization for DROP relative to L_2-Diff-Variant (max).

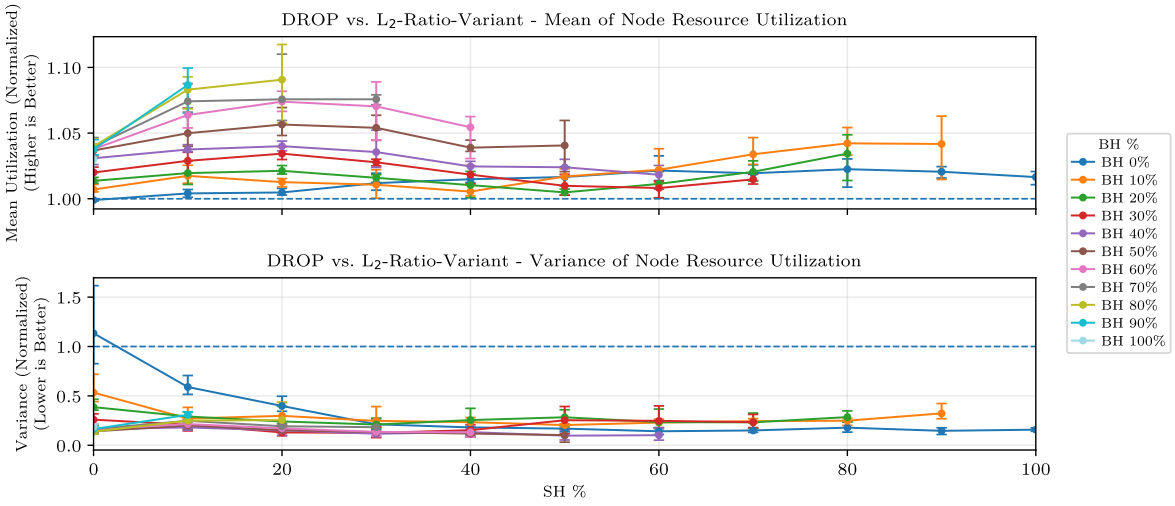

Figure 5.15. Normalized mean and variance of per-node utilization for DROP relative to L_2-Ratio-Variant (min).

5.5 Total Object Acceptance

We now evaluate the placement performance of DROP against the set of baseline heuristics described in Section 5.1. Algorithm performance is measured as the total number of objects successfully placed before the system reaches saturation. We then normalize DROP's performance to each baseline to observe gains achieved. To provide a high-level view of placement algorithm behavior, we first summarize acceptance across workload regimes, then present detailed comparisons across different workload distributions.

We group workloads by regime, ranging from storage-heavy to bandwidth-heavy, with balanced and mixed workloads in between. Figures 5.16 and 5.18 show results for uniform workloads, while Figure 5.17 and 5.19 show results for Pareto workloads.

The x-axis denotes the workload composition, while the y-axis shows the object acceptance of DROP relative to a baseline heuristic. Each line represents a baseline algorithm that DROP is compared against. A y-axis value greater than 1 indicates that DROP accepts more objects than the corresponding baseline. All results are averaged over 5 trials. The error bars indicate the minimum and maximum values observed across trials.

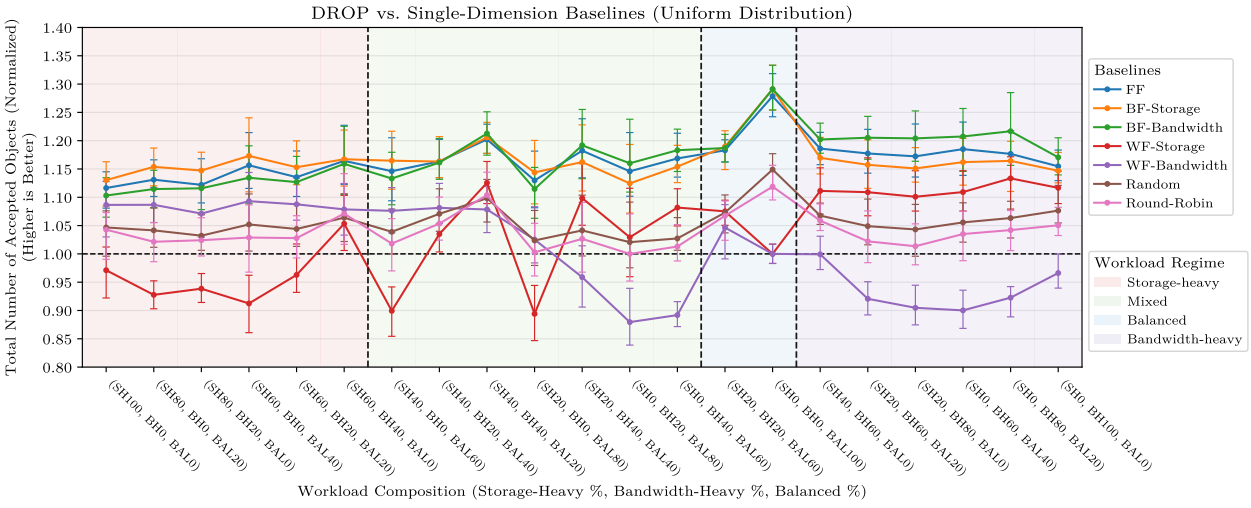

Figure 5.16. Object acceptance of DROP normalized to one-dimensional heuristics across various workload compositions under a uniform demand distribution.

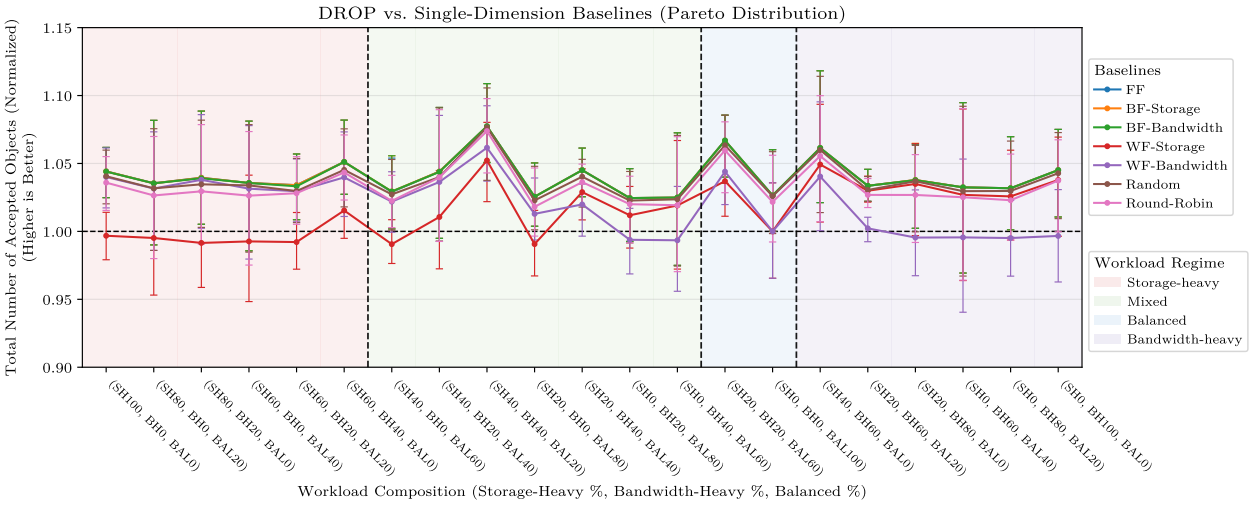

Figure 5.17. Object acceptance of DROP normalized to one-dimensional heuristics across various workload compositions under a Pareto demand distribution.

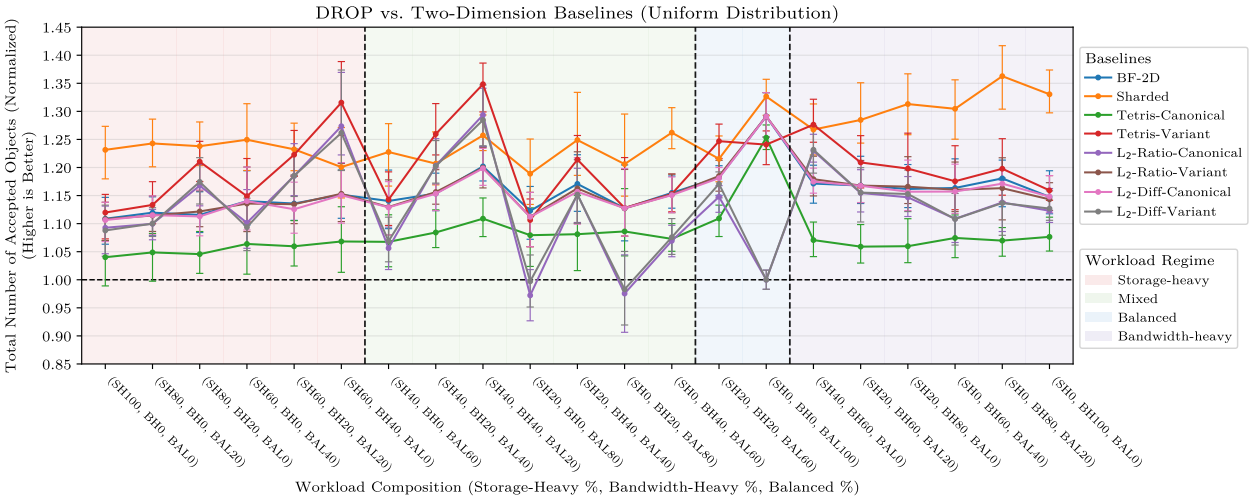

Figure 5.18. Object acceptance of DROP normalized to two-dimensional heuristics across various workload compositions under a uniform demand distribution.

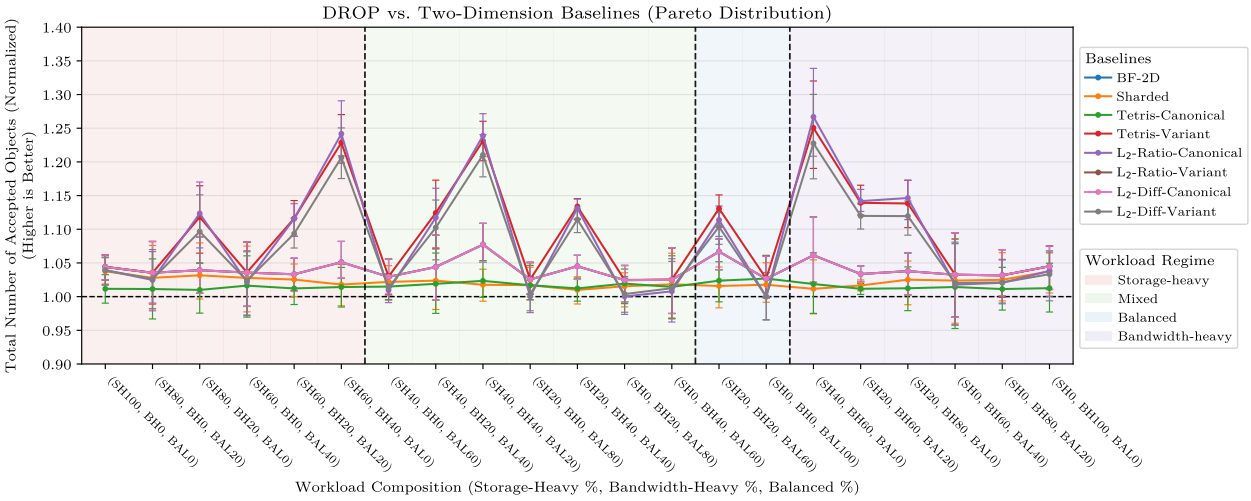

Figure 5.19. Object acceptance of DROP normalized to two-dimensional heuristics across various workload compositions under a Pareto demand distribution.

Across both distributions, several trends can be observed.

First, DROP achieves consistently strong object acceptance across workloads. This indicates that jointly considering storage and bandwidth utilization enables stable performance under varying workload characteristics.

Second, one-dimensional heuristics, namely WF-Storage and WF-Bandwidth, perform well only when the workload aligns with their optimized resource, but degrade as workloads become more mixed. More specifically, these heuristics can outperform DROP in regimes strongly aligned with their optimized resource, but their performance drops when workload characteristics change. We explore this more in Subsection 5.5.1.

Two-dimensional heuristics exhibit different behavior depending on the workload distribution. Under uniform workloads, DROP significantly outperforms all two-dimensional placement algorithms. In contrast, under Pareto workloads, the gap between DROP and other algorithms narrows. In this setting, most objects are small and consume only tiny fractions of node capacity, even when skewed in either dimension. For these objects, differences in placement decisions have minimal impact on system state. The few large objects

that do arrive are highly constrained and have only a limited set of feasible placements. As a result, there is little opportunity for the various algorithms to differentiate.

The First-Fit, Random, and Round-Robin baselines serve as reference points for feasibility-only placement, showing that even when placements are valid, the choice of node significantly impacts long-term acceptance.

The fully sharded baseline removes the inter-node variance by distributing each object across all nodes. However, it does not address the imbalance between storage and bandwidth utilization when the workload itself is skewed, highlighting how performance is limited by workload characteristics rather than placement decisions.

Overall, these results suggest that while some heuristics achieve higher peak performance in specific regimes, DROP provides more consistent performance across diverse workloads. We now focus our detailed analysis on uniform distribution workloads, where placement decisions vary more widely, providing valuable insights into the behavior and performance of the various algorithms.

5.5.1 Comparison Against One-Dimensional Heuristics

We first focus our detailed analysis on one-dimensional heuristics, including First-Fit, Best-Fit (storage and bandwidth), Worst-Fit (storage and bandwidth), Random, and Round-Robin.

Figures 5.20- 5.24 show that DROP consistently achieves higher total object acceptance than First-Fit, Best-Fit (Storage and Bandwidth), Random, and Round-Robin across all workload regimes. These heuristics make placement decisions based on a single dimension or on nonstrategic placement (e.g., ordering in First-Fit or random/round-robin selection) and do not account for utilization in both dimensions. As a result, they frequently exhaust one resource while leaving capacity in the other unused, leading to early infeasibility.

This behavior follows directly from the limitations of one-dimensional placement. By optimizing only a single resource dimension, these heuristics leave capacity in the other underutilized, leading to fragmented node states.

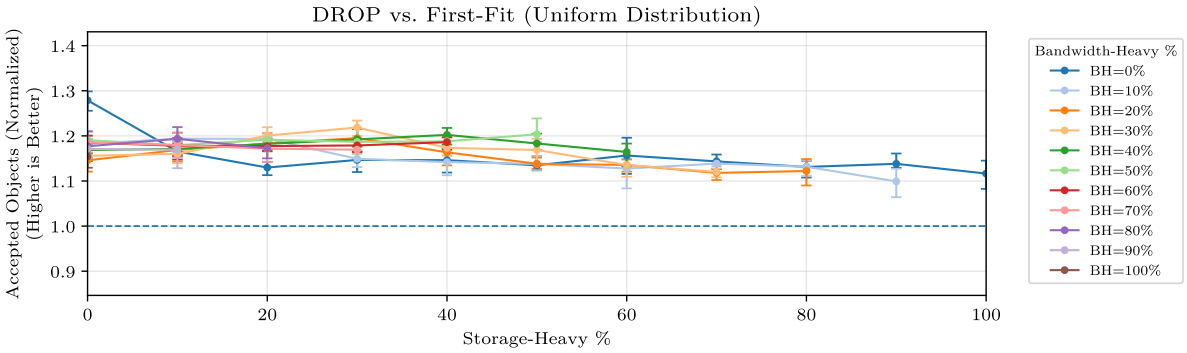

Figure 5.20. Total object acceptance of DROP normalized to First-Fit.

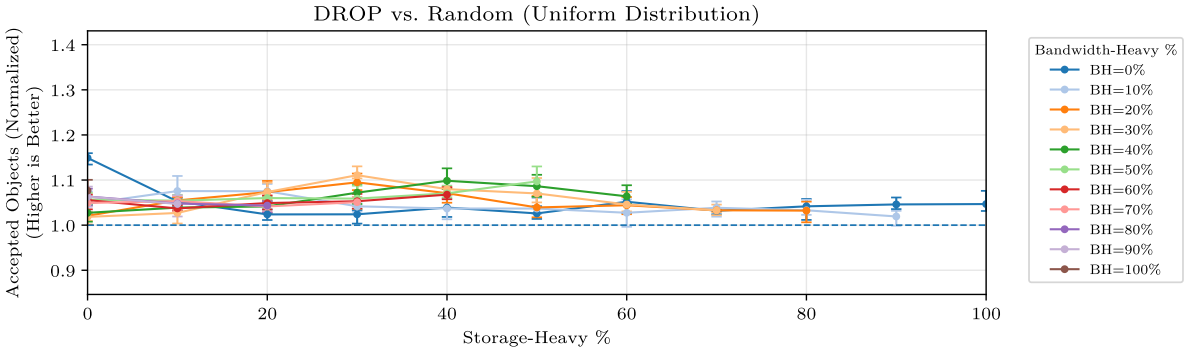

Figure 5.21. Total object acceptance of DROP normalized to random assignment.

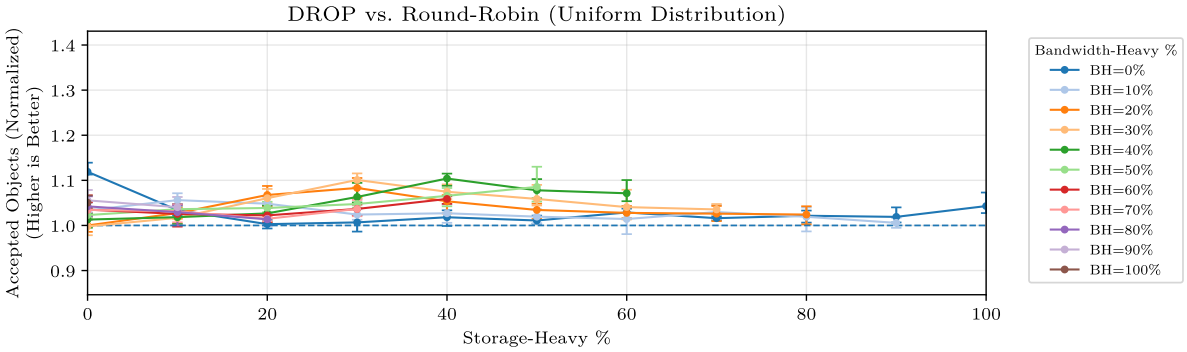

Figure 5.22. Total object acceptance of DROP normalized to Round-Robin assignment.

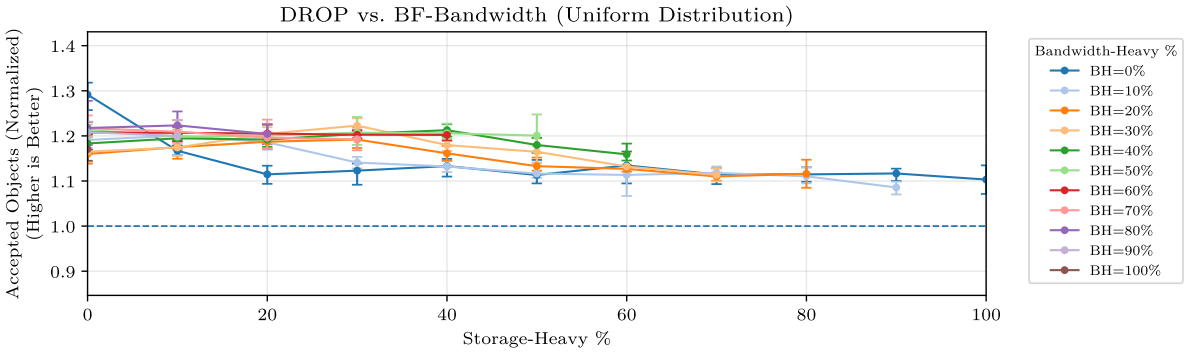

Figure 5.23. Total object acceptance of DROP normalized to BF-Bandwidth.

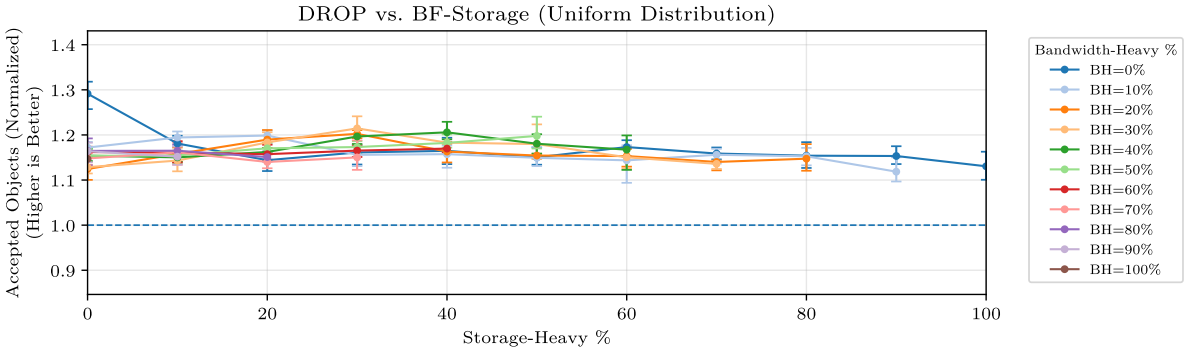

Figure 5.24. Total object acceptance of DROP normalized to BF-Storage.

Worst-Fit heuristics exhibit more nuanced behavior. As shown in Figures 5.25 and 5.26, WF-Bandwidth outperforms DROP when bandwidth-heavy objects dominate, while WF-Storage performs similarly in storage-heavy regimes. However, WF can also outperform DROP in certain regimes that are not purely skewed. In particular, this occurs when a substantial fraction of objects are balanced, but the remaining objects are skewed toward a single resource. Balanced objects constrain both resources but do not bias the system toward either. As a result, the effective bottleneck is determined by the skew in the non-balanced portion of the workload. When this skew is strong, the workload behaves similarly to a one-dimensional setting, allowing the corresponding Worst-Fit heuristic to match or exceed DROP.

Figure 5.27 summarizes this effect across representative workload categories. In storage-heavy and bandwidth-heavy settings, the corresponding one-dimensional heuristic slightly exceeds DROP, reflecting its specialization to that dominant resource. This behavior extends to partially balanced workloads, where the non-balanced objects are strongly skewed toward a single resource. However, as workloads become more mixed and no single resource dominates, both WF-Storage and WF-Bandwidth degrade, while DROP maintains consistently higher acceptance.

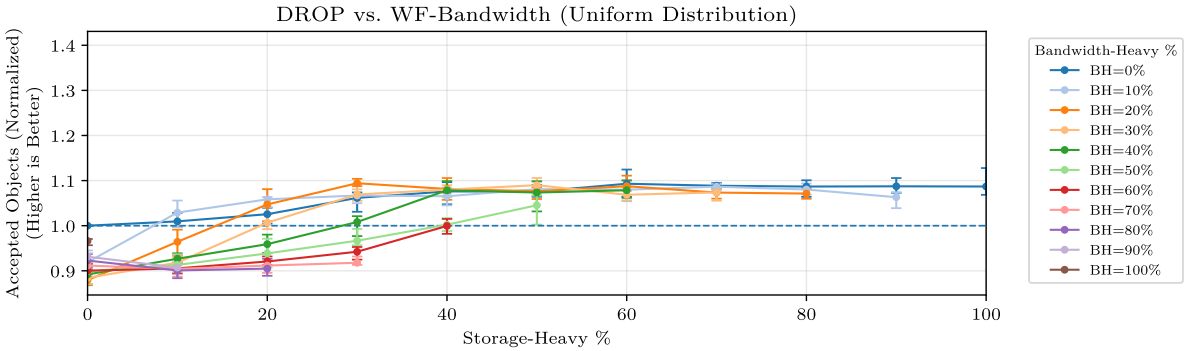

Figure 5.25. Total object acceptance of DROP normalized to WF-Bandwidth.

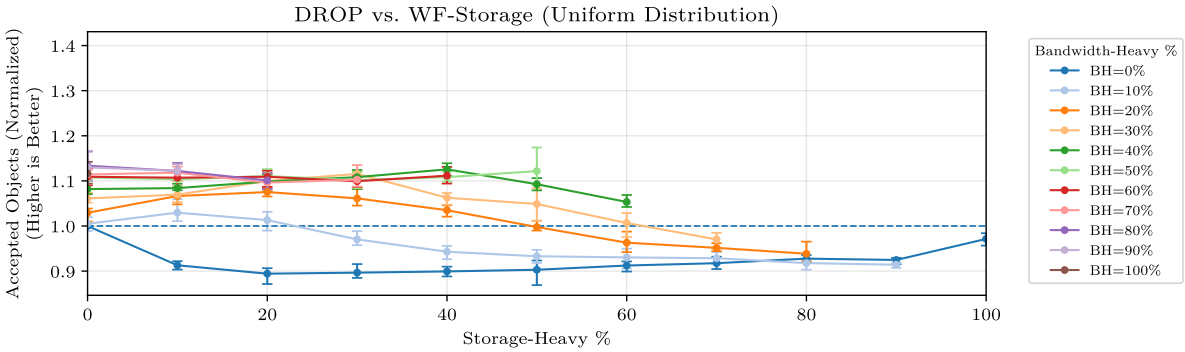

Figure 5.26. Total object acceptance of DROP normalized to WF-Storage.

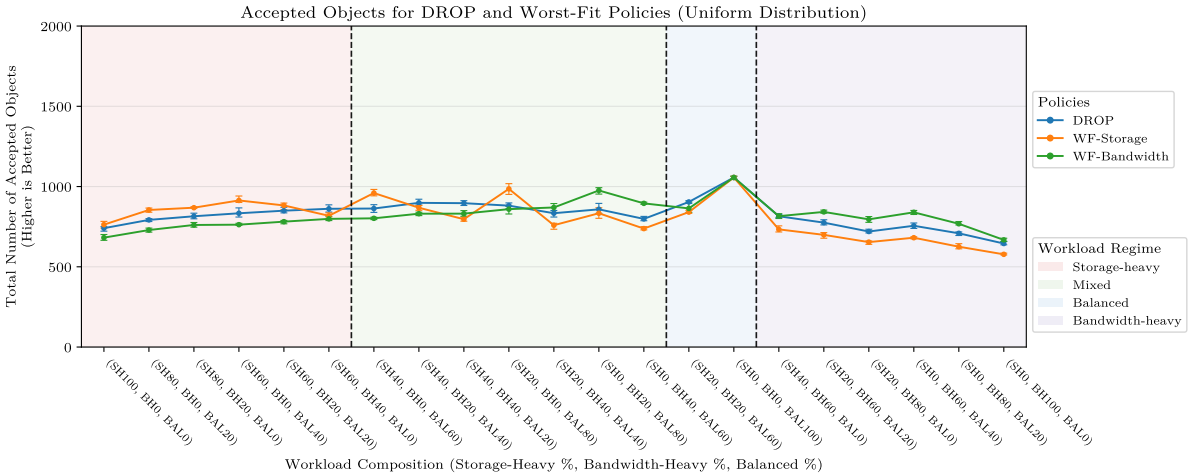

Figure 5.27. Comparison of total accepted objects across representative workload regimes.

Across traces with comparable proportions of storage-heavy, bandwidth-heavy, and balanced objects, we observe that one-dimensional heuristics perform well only when the workload effectively reduces to a single dominant resource, but degrade once both resource constraints must be satisfied simultaneously.

Overall, these results highlight the limitations of one-dimensional heuristics. While they can perform well in regimes aligned with a single resource, they lack robustness across more general workloads. In contrast, DROP maintains consistently strong performance by jointly accounting for both resource dimensions, avoiding the fragmentation that arises from single-dimensional optimization.

We next compare against the two-dimensional heuristics, including BF-2D, the sharded baseline, Tetris, and L_2-based heuristics.

5.5.2 Comparison Against Two-Dimensional Heuristics

Across all of the two-dimensional baseline algorithms (Figures 5.28-5.35), DROP consistently achieves higher total object acceptance. This is evident in mixed workloads, where both storage-heavy and bandwidth-heavy objects are present. In these regimes, most curves remain above 1, indicating that DROP accepts more objects across a wide range of workloads.

Despite considering both resource dimensions, these heuristics are mainly driven by local alignment between object demand and node availability. As a result, they do not control how global node utilization changes, leading to imbalanced node states and fragmented capacity over time.

In contrast, DROP explicitly targets both utilization magnitude and balance, maintaining a more coordinated global placement. This allows the system to sustain higher effective capacity and accept more objects before reaching saturation.

Together with the results for one-dimensional heuristics, this highlights that effective placement requires both multi-dimensional awareness and control over global system state, rather than purely local decision-making.

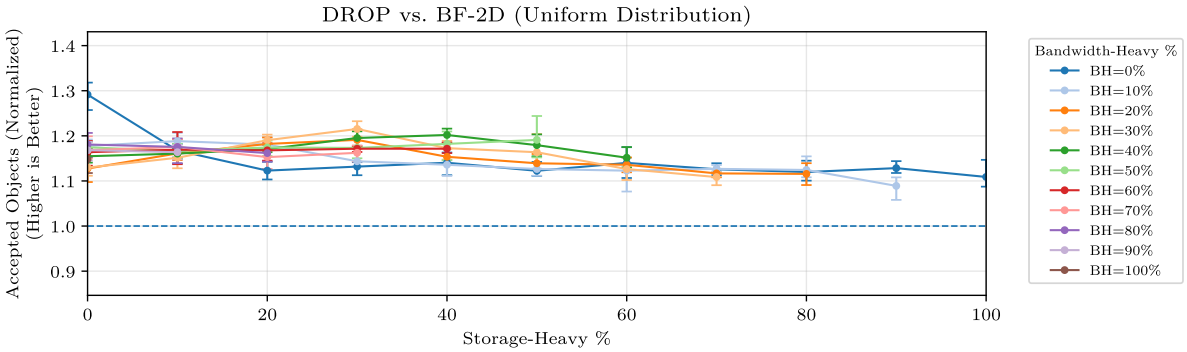

Figure 5.28. Total object acceptance of DROP normalized to BF-2D.

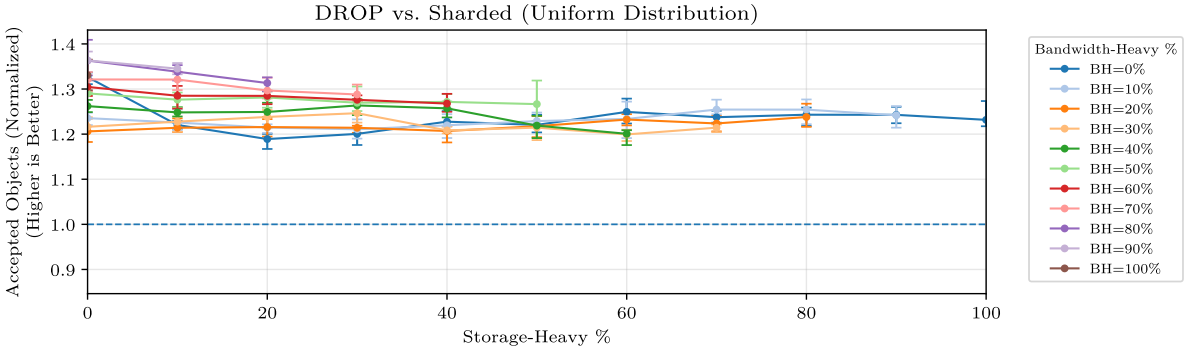

Figure 5.29. Total object acceptance of DROP normalized to naive sharding.

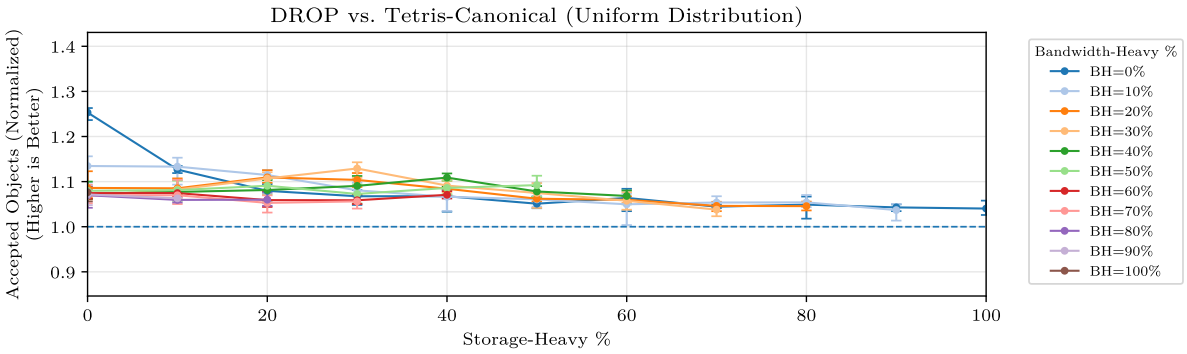

Figure 5.30. Total object acceptance of DROP normalized to Tetris-Canonical.

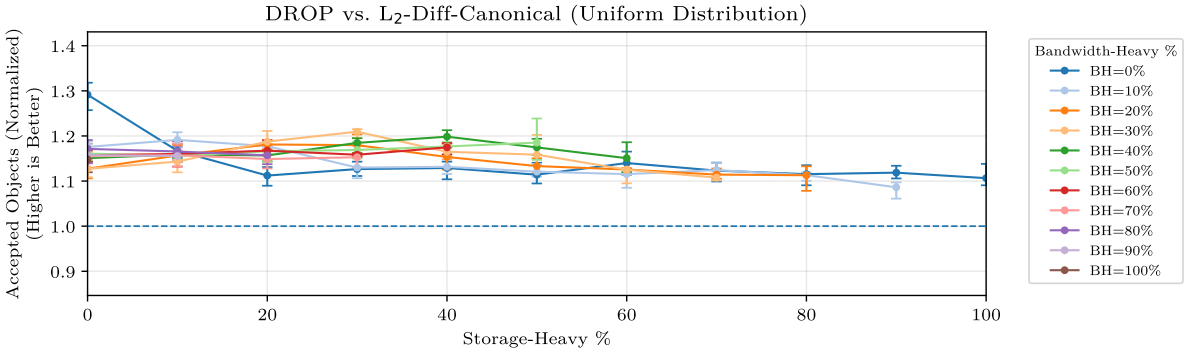

Figure 5.31. Total object acceptance of DROP normalized to L_2-Diff-Canonical.

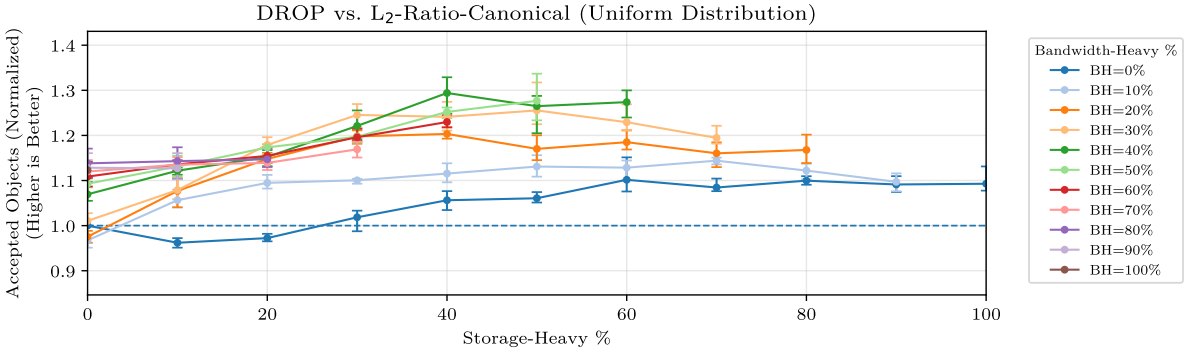

Figure 5.32. Total object acceptance of DROP normalized to L_2-Ratio-Canonical.

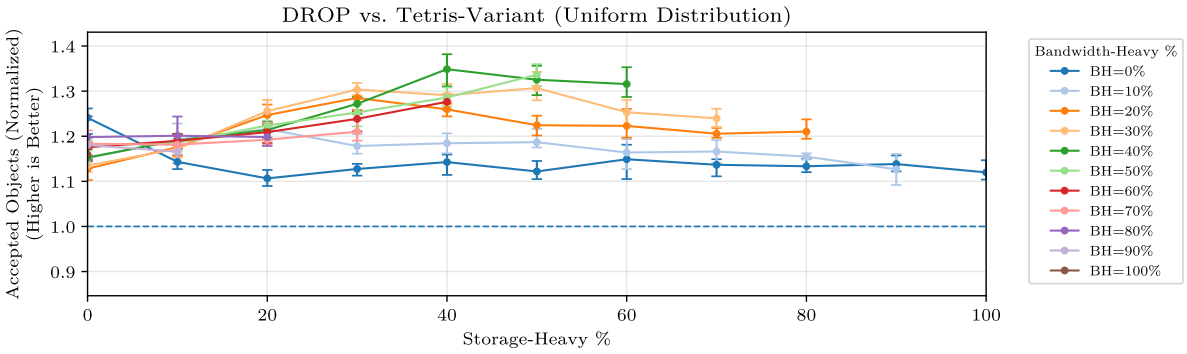

Figure 5.33. Total object acceptance of DROP normalized to a variant of Tetris where the least-aligned node is selected.

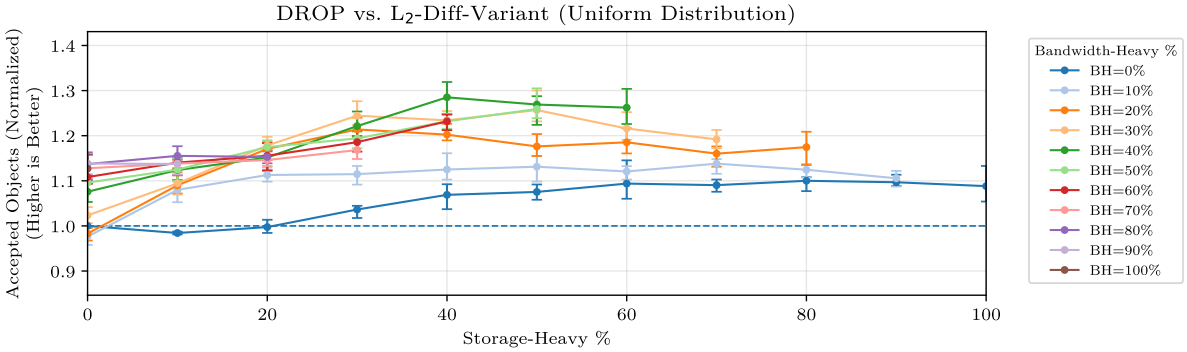

Figure 5.34. Total object acceptance of DROP normalized to a variant of L_2-Diff where the least-aligned node is selected.

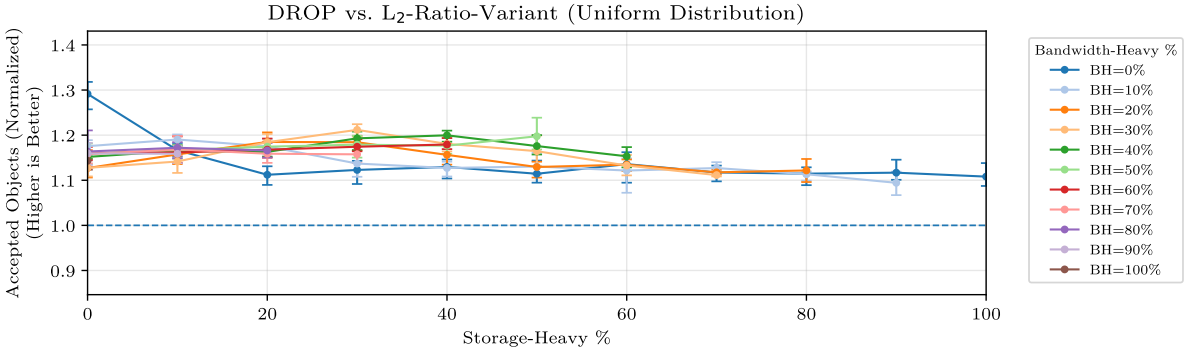

Figure 5.35. Total object acceptance of DROP normalized to a variant of L_2-Ratio where the least-aligned node is selected.

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

This thesis studies the problem of online object placement in disaggregated systems with storage and network bandwidth constraints. Unlike traditional bin-packing formulations, which consider a single resource dimension, this setting requires reasoning about interactions among multiple resources under sequential, irreversible placement decisions. The objective is to maximize the total number of objects that can be accepted before the system reaches saturation.

This thesis introduced DROP, a placement heuristic that jointly accounts for utilization magnitude and direction across resource dimensions. Rather than optimizing a single dimension or relying on local alignment between object demand and memory node availability, DROP explicitly aims for the globally optimal object placement by guiding node utilization towards balance.

Through extensive evaluation, we demonstrated that this design yields two key outcomes. First, DROP provides a more concentrated distribution of node states, reducing variance and avoiding the formation of imbalanced nodes that prematurely exhaust a single resource. Second, this improved utilization translates into higher total object acceptance across a wide range of workload compositions. While certain baselines perform well in highly specific regimes, none maintain strong performance across the full spectrum of workloads. In contrast, DROP consistently delivers strong performance, demonstrating robustness across varying workloads.

6.2 Future Work

Several directions remain for continuing this work.

First, this thesis assumes that object resource demands are known at placement time. In practice, bandwidth demands may need to be estimated dynamically and may vary over time. Extending DROP to operate under uncertainty or time-varying demand is an important direction. Predicting future demands based on historical or network characteristics [23] is also a promising direction for aiding DROP's object placement algorithm.

Second, integrating DROP into a practical system is a natural direction for the future. This will require determining which components of DROP's algorithm are implemented where, e.g., at the switch [24, 25], at the compute node [26], or at the memory node.

Third, the current formulation constrains placement to a single memory node. In practice, a larger object can be broken down into page-sized chunks [27] and distributed across multiple memory nodes. Such sharding can improve balance across both storage and bandwidth, but it introduces additional complexity, including coordination across nodes, and increased network traffic during access. Understanding when sharding provides a net benefit and how to incorporate it into DROP's placement algorithm remains an open question.

Fourth, the current formulation assumes that placement decisions are irrevocable. While this reflects the practicality of DROP, allowing limited re-balancing or migration could further improve system utilization. Investigating how to incorporate re-placement or periodic migrations is a natural extension.

Finally, this evaluation is conducted in a simulation environment with controlled workloads. Applying DROP in real systems would help validate its effectiveness under real conditions.

REFERENCES

[1] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, MA: USENIX Association, Mar. 2017, pp. 649–667, ISBN: 978-1-931971-37-9. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu.

[2] Z. Ruan, M. Schwarzkopf, M. K. Aguilera, and A. Belay, "AIFM: High-Performance, Application-Integrated far memory," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*, USENIX Association, Nov. 2020, pp. 315–332, ISBN: 978-1-939133-19-9. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/ruan.

[3] E. Amaro *et al.*, "Can far memory improve job throughput?" In *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20, Heraklion, Greece: Association for Computing Machinery, 2020, ISBN: 9781450368827. DOI: 10.1145/3342195.3387522. [Online]. Available: https://doi.org/10.1145/3342195.3387522.

[4] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "LegoOS: A disseminated, distributed OS for hardware resource disaggregation," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Carlsbad, CA: USENIX Association, Oct. 2018, pp. 69–87, ISBN: 978-1-939133-08-3. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/shan.

[5] W. Su and V. Shrivastav, "Edm: An ultra-low latency ethernet fabric for memory disaggregation," in *30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '25)*, 2025.

[6] D. S. Johnson, "Near-optimal bin packing algorithms," Ph.D. dissertation, Massachusetts Institute of Technology, 1973. [Online]. Available: https://dspace.mit.edu/handle/1721.1/57819.

[7] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," Jan. 2011. [Online]. Available: https://www.microsoft.com/en-us/research/publication/heuristics-for-vector-bin-packing/.

[8] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14, Chicago, Illinois, USA: Association for Computing Machinery, 2014, pp. 455–466, ISBN: 9781450328364. DOI: 10.1145/2619239.2626334. [Online]. Available: https://doi.org/10.1145/2619239.2626334.

[9] Memcached Project, *Memcached: A distributed memory object caching system*, https://memcached.org/, Accessed: 2026-04-12.

[10] J. C. Corbett *et al.*, "Spanner: Google's globally-distributed database," in *OSDI*, 2012.

[11] G. DeCandia *et al.*, "Dynamo: Amazon's highly available key-value store," in *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07, Stevenson, Washington, USA: Association for Computing Machinery, 2007, pp. 205–220, ISBN: 9781595935915. DOI: 10.1145/1294261.1294281. [Online]. Available: https://doi.org/10.1145/1294261.1294281.

[12] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, {*FaRM*}*: Fast Remote Memory*. NSDI, 2014.

[13] Mellanox Technologies Ltd., "ConnectX-5 Ethernet Adapter Cards," NVIDIA Networking, Tech. Rep., 2020, Datasheet. [Online]. Available: https://network.nvidia.com/files/doc-2020/pb-connectx-5-en-card.pdf.

[14] Wikipedia, *DDR5 SDRAM*, Last accessed April 19, 2026. [Online]. Available: https://en.wikipedia.org/wiki/DDR5_SDRAM.

[15] C. Wang *et al.*, "Canvas: Isolated and adaptive swapping for Multi-Applications on remote memory," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, Boston, MA: USENIX Association, Apr. 2023, pp. 161–179, ISBN: 978-1-939133-33-5. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/wang-chenxi.

[16] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11, Boston, MA: USENIX Association, 2011, pp. 323–336.

[17] M. Tirmazi *et al.*, "Borg: The next generation," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20, Heraklion, Greece: Association for Computing Machinery, 2020, ISBN: 9781450368827. DOI: 10.1145/3342195.3387517. [Online]. Available: https://doi.org/10.1145/3342195.3387517.

[18] B. Hindman *et al.*, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11, Boston, MA: USENIX Association, 2011, pp. 295–308.

[19] I. Gog, M. Schwarzkopf, A. Gleave, R. N. M. Watson, and S. Hand, "Firmament: Fast, centralized cluster scheduling at scale," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA: USENIX Association, Nov. 2016, pp. 99–115, ISBN: 978-1-931971-33-1. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gog.

[20] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001. DOI: 10.1109/71.963420.

[21] Y. Lee, H. A. Maruf, M. Chowdhury, A. Cidon, and K. G. Shin, "Hydra : Resilient and highly available remote memory," in *20th USENIX Conference on File and Storage Technologies (FAST 22)*, Santa Clara, CA: USENIX Association, Feb. 2022, pp. 181–198, ISBN: 978-1-939133-26-7. [Online]. Available: https://www.usenix.org/conference/fast22/presentation/lee.

[22] V. V. Vazirani, *Approximation Algorithms*. Springer, 2001, Available online. [Online]. Available: https://www.ics.uci.edu/~vazirani/book.pdf.

[23] J. Lei and V. Shrivastav, "Seer: Enabling future-aware online caching in networked systems," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI '24)*, 2024.

[24] V. Shrivastav, "Stateful multi-pipelined programmable switches," in *36th ACM Special Interest Group on Data Communication Conference (SIGCOMM '22)*, 2022.

[25] V. Shrivastav, "Programmable multi-dimensional table filters for line rate network functions," in *36th ACM Special Interest Group on Data Communication Conference (SIGCOMM '22)*, 2022.

[26] D. Firestone *et al.*, "Azure accelerated networking: Smartnics in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*, 2018.

[27] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 9th. Wiley Publishing, 2012, ISBN: 1118063333.