# Narwhal: Breaking the Local Boundary via Disaggregated Memory Scheduling for Alibaba AnalyticDB

Jian Zhou*
Jiachi Zhang*
Yang Zhang
Alibaba Cloud
Hangzhou, China

Yongbo Wang
Ruonan Guo
Yuemin Wu
Alibaba Cloud
Hangzhou, China

Zeguang Chang
Shuai Wang
Ye Yin
Alibaba Cloud
Hangzhou, China

Wenchao Zhou
Liang Lin
Alibaba Cloud
Hangzhou, China

{jan.zhouj,zy334083,zhangjiachi.zjc,wangyongbo.wyb,grace.grn,}
{yuemin.wym,changzeguang.czg,shuaiquan.ws,yinye.yin,}
{zwc231487,yibo.ll}@alibaba-inc.com

## ABSTRACT

Cloud OLAP workloads are bursty and memory-hungry. Per-machine DRAM caps and co-scaled CPU/memory provisioning are fragile. They lead to low utilization, slow autoscaling, and transient OOMs. Prediction-driven schedulers and bin-packing optimizers mitigate but struggle with rigid per-machine memory boundaries and forecasting errors, Existing RDMA or OS-level remote-memory approaches either assume single-tenant trusted clusters (lacking hardware isolation and QoS) or treat remote DRAM as opaque swap. We present Narwhal, an RDMA-co-designed analytical database. It jointly schedules local DRAM and a shared remote memory pool to trade consolidation against bounded remote borrowing. It enforces hardware-level multi-tenant isolation with per-tenant RDMA Protection Domains, quotas, and QoS. It uses a database-aware memory allocator with a three-tier cache and a pressure-threshold invariant to avoid OOMs and protect metadata. Integrated into a production engine, Narwhal eliminates OOMs, boosts memory overselling from 92.67% to **121.89%** (**+29.22**) and node utilization from 78.16% to **96.63%** (**+18.47**), while limiting average remote memory borrowing to just **1.63%**.

**PVLDB Reference Format:**
Jian Zhou, Jiachi Zhang, Yang Zhang, Yongbo Wang, Ruonan Guo, Yuemin Wu, Zeguang Chang, Shuai Wang, Ye Yin, Wenchao Zhou, and Liang Lin. Narwhal: Breaking the Local Boundary via Disaggregated Memory Scheduling for Alibaba AnalyticDB. PVLDB, 19(12): 4466 - 4479, 2026.
doi:10.14778/3827998.3828046

## 1 INTRODUCTION

Cloud data warehouses increasingly run in elastic, multi-tenant environments where improving resource utilization is central to lowering cost under stringent SLOs. Prior industrial and academic systems pursue efficiency by predicting workload demand and improving machine-level bin packing, including autoscaling in Borg [41] and Kubernetes [19], Autopilot-style policies [34, 35], CherryPick [33], and multidimensional bin packing in Eigen/Eigen+ [25, 26].

However, these approaches face two fundamental limitations when confronted with bursty, memory-intensive OLAP workloads. First, the **"rigid boundary"** of per-machine resources makes accurate prediction insufficient. When a memory spike occurs, local CPU and memory cannot elastically expand, causing transient contention or slow autoscaling and, in the worst case, out-of-memory (OOM) failures that violate SLOs [3, 21, 25]. Production experience confirms that even small forecasting errors are amplified at high utilization, and that a 100% correct approximation of future demand is not achievable [4, 18, 48, 49]. Second, most prior work assumes tightly coupled, **co-scaled CPU and memory**. Fixed-proportion scaling leads to chronic resource mismatch, where the least-capacity dimension becomes the bottleneck and the remaining dimensions are wasted [9, 36].

To overcome single-machine bottlenecks, recent work adopts resource disaggregation: compute, memory, and storage are physically decoupled and offered as independently scalable pools that can be composed on demand [9, 36]. Figure 1 illustrates two representative lines of prior art. Panel (a) depicts systems that run single-tenant databases over remote memory accessed via high-performance interconnects such as RDMA (e.g., FaRM [8], Mira [13], ROLEX [50], and HERD [14]). Panel (b) highlights disaggregated cloud databases that separate compute from shared memory or storage, including PolarDB Serverless [52] and the disaggregated shared-memory design PolarDB-MP [53].

Yet, when applied to multi-tenant cloud data warehouses, existing solutions leave three challenges unaddressed.

**C1: Lack of jointly scheduling of local and remote resources.** Most systems treat remote memory as a simple extension of local memory and lack a first-class model for jointly scheduling heterogeneous resources. In the absence of coordinated quota allocation, placement, and admission control, the system cannot decide *when* and *how much* remote memory to provision per tenant to minimize total cost while bounding the frequency of remote access. Without such coordination, the system cannot safely oversubscribe memory (i.e., allocate beyond physical capacity via statistical multiplexing), leaving the potential efficiency gains of disaggregation unrealized.

**C2: Weak multi-tenant isolation.** Many RDMA-based remote memory systems are designed for trusted cluster environments [8,

*Both authors contributed equally to this research.


Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828046

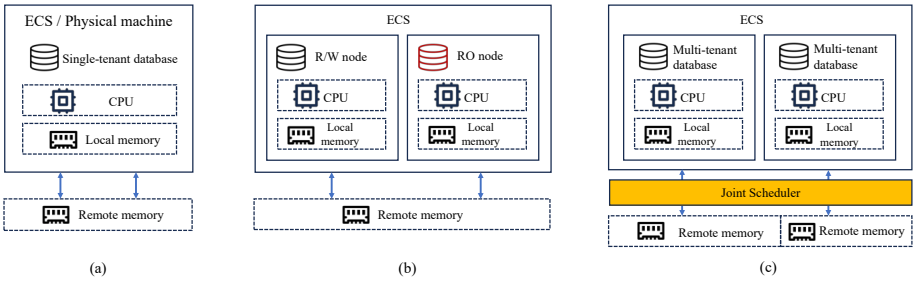



**Figure 1: Disaggregated architectures: (a) single-tenant database backed by remote memory, (b) multi-tenant database backed by single-tenant remote memory, and (c) multi-tenant database backed by multi-tenant remote memory**

14, 16, 27, 30, 44, 45]. They typically share a single RDMA Protection Domain across all processes or rely on software-level access checks, providing no hardware boundary between tenants and leaving the system vulnerable to noisy-neighbor interference [37]. Database-integrated systems provide instance-level isolation but are too coarse to protect workers within the same instance competing for shared remote memory.

**C3: OLAP workload mismatch.** Most remote memory studies are tuned for transactional or key–value access patterns [8, 14, 16, 44, 45]. Directly sinking all cache blocks to a remote pool incurs unnecessary network traffic for scan-heavy OLAP workloads. Failing to recognize OLAP-specific data semantics (e.g., the distinction between latency-critical metadata blocks and data blocks) causes metadata thrashing and bandwidth waste.

We argue that multi-tenant cloud data warehouses need a third architecture beyond "**single-tenant database backed by remote memory**" from Figure 1 (a), and "**multi-tenant database backed by single-tenant remote memory**" from Figure 1 (b). The target is "**multi-tenant database backed by multi-tenant remote memory**" from Figure 1 (c), with joint optimization of the use of local and remote memory, multi-tenant isolation, and OLAP-specific optimizations.

A straightforward idea is to reuse OS-level remote memory management. Swap-based systems such as Infiniswap [11], Fastswap [1], and Leap [28], and cluster-wide pooling systems such as Spirit [23], Pond [12], and Canvas [42] provide increasingly sophisticated paging, prefetching, and fault-tolerance mechanisms. However, these approaches are insufficient for multi-tenant analytical databases: they perform no cross-tenant cost-aware scheduling (**C1**), do not extend isolation to the RDMA data path (**C2**), and manage memory at page granularity without awareness of application-level data semantics (**C3**). Experience in database systems confirms that treating memory management as transparent to the database sacrifices significant optimization opportunities [2, 39].

This paper presents Narwhal, an RDMA-co-designed cloud-native analytical database (Alibaba AnalyticDB) that, to our knowledge, is the **first system to treat remote memory as a first-class scheduling dimension** in a multi-tenant OLAP engine. Neither prior oversubscription schedulers nor prior RDMA remote memory systems jointly optimize local and remote capacity: the former are confined within rigid per-machine boundaries, and the latter expose remote memory as a passive transparent pool with no cross-tenant cost-aware quota allocation. Narwhal's borrowing-ratio formulation explicitly decides when and how much each worker borrows from the remote pool, transforming memory spikes from hard OOM failures into soft overload and raising overall utilization.

First, Narwhal introduces a **joint scheduler** that treats remote memory as a first-class scheduling dimension. It minimizes TCO across local and remote resources under tenant SLO and capacity constraints, determining per-worker memory quotas and safe oversubscription levels. Second, Narwhal enforces **hardware-level multi-tenant isolation** via RDMA Protection Domains. Each tenant's QPs, MRs, and CQs are bound to a distinct PD; the NIC rejects cross-PD access in hardware, so even a malicious tenant holding another's rkey cannot access its data. This mechanism also isolates worker groups within the same database instance. Third, Narwhal employs an **OLAP-aware Memory Allocator** (Section 6) with a three-tier hierarchy (local DRAM / remote memory / disk). It dynamically migrates blocks under a pressure-threshold invariant that prevents OOM during transient spikes, and uses type-aware placement to keep latency-critical metadata blocks local while sinking bulk data blocks to remote memory.

In summary, this work makes the following contributions.

- A **joint scheduler** that treats remote memory as a first-class scheduling dimension, using a remote-borrowing-ratio formulation to safely oversubscribe memory beyond per-machine capacity—the first system to do so for multi-tenant OLAP. Hardware-enforced PD isolation and OLAP-aware data placement serve as necessary enablers.
- Production deployment in AnalyticDB, demonstrating that Narwhal boosts the memory overselling rate from 92.67% to **121.89%** (**+29.22**) and node utilization from 78.16% to **96.63%** (**+18.47**), without OOM incidents.

## 2 BACKGROUND

### 2.1 Resource Scheduling

Resource scheduling in modern cloud systems is commonly formulated as a $d$-dimensional vector bin packing ($d$-VBP) problem), where each machine is a bin with capacity vector and each workload is an item with multi-resource demand (e.g., CPU, memory, and I/O) [26].

In practice, when one resource becomes the dominant bottleneck (often memory for database workloads), the scheduling objective can be reduced to a one-dimensional bin packing problem over that bottleneck resource: Given a set of database instances $I_1, I_2, ..., I_n$ and a set of machines $M_1, M_2, ..., M_m$, each instance $I_i$ has a memory

allocation $A_i$ and a memory utilization time series $U_i(t)$. The goal is to place instances onto machines while minimizing the number of machines, subject to a safety threshold $T \in [0, 1]$ that bounds machine-level aggregated utilization: $\max_t \sum_{i \in M_j} U_i(t) \leq T,\ \forall M_j$.

If the future utilization were known, the optimal scheduler would directly pack according to the true future utilization series $\widehat{U}_i(t)$ for each instance and enforce the above constraint exactly. In reality, $\widehat{U}_i(t)$ is unknown, so practical systems approximate it with an estimated item size $\widetilde{U}_i$ or an estimated series $\widetilde{U}_i(t)$ derived from historical observations, and then run bin packing under the corresponding approximate constraint.

Prior work differs mainly in how $\widehat{U}$ is estimated from observed $U(t)$, spanning percentile-based [4, 34], stochastic [4, 6, 43, 48], Monte Carlo [18], and predictive [20, 31, 38, 40, 54] families. Eigen+ [25] couples such estimators with bin packing by classifying instances as steady or transient and assigning conservative item sizes to the latter, improving utilization while bounding OOM risk.

However, the fundamental challenge remains that a 100% correct approximation of $\widehat{U}_i(t)$ is not practical in production: workload shifts and spikes can invalidate statistical assumptions, forecasting errors are unavoidable, and even small errors can cause constraint violations at high utilization, leading to OOM events and SLO degradation [3, 21].

## 2.2 RDMA-Based Remote Memory.

Resource disaggregation physically decouples compute, memory, and storage into independently scalable pools that can be composed on demand [9, 36]. Remote Direct Memory Access (RDMA) allows a process on one machine to read or write the memory of another machine without involving the remote CPU, achieving single-digit microsecond latency and near-line-rate throughput [15].

The RDMA programming model centers on several key abstract [15, 32]: a **Queue Pair** (QP) carries work requests between a consumer and a provider; a **Memory Region** (MR) is a contiguous buffer registered with the NIC and identified by a remote key (`rkey`) that authorizes access; a **Protection Domain** (PD) groups QPs and MRs so that only QPs within the same PD may access the associated MRs. RDMA verbs (`READ`, `WRITE`) bypass the remote CPU entirely, making them attractive for cache-tier and memory-pool architectures where data can be fetched without waking a remote server thread.

A central motivation for RDMA-based disaggregation is the growing imbalance between memory capacity and compute density in modern clusters: many machines have underutilized DRAM while others are memory-constrained [9, 11, 36]. Exposing idle DRAM on remote machines as an elastic memory tier bridges the latency gap between local DRAM (~100 ns) and SSD (~100 $\mu$s). Remote memory access via RDMA typically completes in 2–5 $\mu$s [8, 11, 15]—roughly 20–50× slower than local DRAM but 20–50× faster than SSD. Each PACE server is equipped with a 100 Gbps NIC, yielding ~12.5 GB/s effective bandwidth per node. This is below a multi-disk VM configuration (~40 GB/s aggregate SSD bandwidth), but remote-memory bandwidth is not the bottleneck in Narwhal's deployment, for three reasons. *First, low remote-borrowing ratio:* production measurements (Section 7.1) show that remote borrowing accounts for only 1.63% of each pod's aggregate memory usage on average (max 2.90%); since RDMA bandwidth demand scales with the volume of remote-resident data, this bounds time-averaged bandwidth per pod well below 1 Gbps. *Second, type-aware placement:* only latency-critical metadata blocks (Index/Filter) traverse the RDMA path (Section 6.2); bulky data blocks and sequential scans continue at local SSD bandwidth, so the 40 GB/s aggregate SSD path is preserved for bandwidth-heavy OLAP traffic. *Third, elastic scaling:* higher aggregate remote bandwidth is obtained by provisioning additional PACE servers independently of compute nodes. The pressure experiment in Figure 7e further validates that small-request latency remains stable even when competing tenants saturate the 100 Gbps link. This positioning makes remote memory an attractive intermediate tier for caching warm data that would otherwise be evicted to disk, an observation that has been validated in both academic prototypes [1, 11, 28] and production cloud systems [13, 22, 24].

Two architectural patterns have emerged for exploiting this tier. In the *application-integrated* pattern, the database or key–value store is explicitly redesigned to issue RDMA operations as part of its data path. FaRM [8], DrTM [45], Pilaf [30], HERD [14], etc. co-design data layout and access protocols with RDMA verbs to build transactional stores, indexes, and shared-memory abstractions that minimize round trips. In the *OS-transparent* pattern, remote DRAM is exposed as a swap device, page-fault handler, or cluster-wide memory pool, allowing applications to benefit from remote memory without code changes. Fastswap [1], Leap [28], Carbink [55], Spirit [23], Pond [12], and Canvas [42] are representative systems in this family, providing increasingly sophisticated paging, prefetching, fault tolerance, and pooling mechanisms at the OS or runtime level, but uniformly treating remote memory as semantically opaque to the application.

Despite the rich design space, existing RDMA-based remote memory systems face security limitations when applied to multi-tenant cloud data warehouses. On the one hand, application-integrated systems universally assume a single-tenant, trusted cluster: they typically share a single Protection Domain across all processes or rely on software-level access checks, providing no hardware boundary between tenants and leaving the system vulnerable to noisy-neighbor interference. On the other hand, OS-transparent systems inherit process- or VM-level isolation from the kernel but do not extend it to the RDMA data path; once a memory region is registered, any QP within the same PD can access it regardless of tenant identity.

## 2.3 Memory Management in AnalyticDB

AnalyticDB [7, 51] is Alibaba Cloud's cloud-native OLAP database service. Each AnalyticDB instance consists of a set of *workers*, each of which manages a partition of the data and executes query fragments in parallel. Internally, each worker uses an LSM-tree–based storage engine to persist columnar data on SSD or distributed file systems.

The memory consumed by an AnalyticDB worker falls into two broad categories based on how it is managed:

(1) **Bounded cache memory** The primary consumer is the *block cache*, a hash-table–indexed in-memory cache that holds three types of blocks read from SST files:
   - `Data Blocks`: compressed columnar data segments;

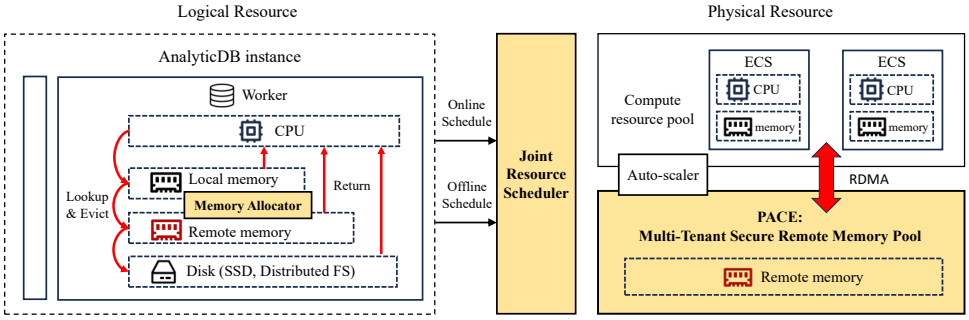



**Figure 2: Overview of NARWHAL**

- `Index Blocks`: per-SST B-tree–style index entries that locate `Data Blocks` by key range;
- `Filter Blocks`: Bloom filters used to skip SSTables that cannot contain a queried key.

These blocks are inserted into the cache upon read and evicted under a unified replacement policy (e.g., LRU or Clock) when the cache reaches its configured capacity. Because this memory is managed through a hash table with explicit eviction, its footprint is *bounded and controllable*: the system can shrink or grow the cache by adjusting the capacity limit and evicting entries accordingly.

(2) **Transiently allocated memory.** The remaining memory is allocated on demand by query operators and internal subsystems, including intermediate buffers for hash joins, aggregations, and sorts; write buffers (`MemTable`) for LSM-tree ingestion; network send/receive buffers; and thread stacks, metadata structures, and other runtime overhead. This memory is requested from the OS allocator (e.g., `malloc`/`jemalloc`) when needed and released when the operation completes. Its footprint is *demand-driven and difficult to predict*: a single large join or a burst of concurrent queries can cause a transient spike that temporarily exhausts available DRAM.

In production, the interplay between these two categories creates a tension. Cache-managed memory is bounded but consumes a large share of local DRAM to maintain acceptable hit rates. Transient allocations are unbounded and bursty. When a transient spike occurs, the OS may not be able to satisfy the allocation because the block cache still holds its budgeted capacity, leading to out-of-memory (OOM) failures. Shrinking the cache proactively wastes capacity during normal operation; shrinking it re-actively may be too slow to prevent OOM under sudden bursts. This motivates the *Memory Allocator* component introduced in Section 3, which coordinates local and remote memory to provide elastic headroom for transient allocations without sacrificing cache performance.

## 3 OVERVIEW

This section presents the architecture of NARWHAL and explains how its components address the three challenges identified in Section 1: lack of joint scheduling of local and remote resources (C1), weak multi-tenant isolation (C2), and OLAP workload mismatch (C3).

Figure 2 illustrates the architecture of NARWHAL, organized along two dimensions: a *logical resource* view (left) that captures the database engine's perspective, and a *physical resource* view (right) that captures the underlying infrastructure.

On the *logical side*, each AnalyticDB instance contains one or more workers. Each worker has access to four resource types: CPU for query execution, *local memory* (DRAM on the compute node) for caching and transient allocations, *remote memory* accessible via RDMA for elastic cache extension, and *disk* (SSD or distributed file system) for persistent storage. The **Memory Allocator** (Section 6) sits between each worker and the local/remote memory tiers. As shown in Figure 2, it intercepts lookup and eviction operations and organizes the three storage tiers into a coherent hierarchy: local DRAM serves as L1, remote memory as L2, and disk as L3. When a lookup hits a block cached in remote memory, the Memory Allocator fetches it via RDMA `READ` and returns it to the worker transparently. When local memory pressure rises, it evicts victim blocks to remote memory via RDMA `WRITE` rather than dropping them to disk, maintaining the pressure-threshold invariant described in Section 6.1.

On the *physical side*, compute nodes reside in an ECS-based compute resource pool, where each ECS instance provides co-located CPU and local DRAM. A separate **PACE** (Section 5) cluster provides the multi-tenant secure remote memory pool. PACE partitions its aggregate DRAM into per-tenant data regions, each protected by a distinct RDMA Protection Domain, and enforces capacity quotas, weighted fair queuing, and bandwidth throttling to prevent noisy-neighbor interference. A dedicated PACE cluster avoids a circular dependency where workers needing remote memory during spikes are themselves at peak occupancy and cannot contribute idle DRAM to a shared pool. Compute nodes access PACE over a RoCEv2 RDMA fabric, achieving 2–5 $\mu$s round-trip latency for remote cache lookups.

The **Joint Resource Scheduler** (Section 4) bridges the logical and physical views. It takes as input the set of workers, their observed and predicted memory demand time series, and the capacities of compute nodes and the remote memory pool. It produces two outputs: a placement of workers onto compute nodes (determining how local DRAM is partitioned among co-located workers) and per-worker local quotas that control the boundary between local and remote memory usage. The scheduler operates in two modes. *Online scheduling* (Section 4.2.1) handles incremental events—instance creation, scale-out, worker restart—by performing best-fit bin packing over local DRAM capacity using declared reservation requests, without requiring demand forecasts. *Offline*

*scheduling* (Section 4.2.2) runs periodically to recompute local quotas based on updated demand predictions, repack workers across machines to improve consolidation under a migration budget, and estimate the remote memory capacity required to sustain predicted peak borrowing. An auto-scaler adjusts the compute pool size based on scheduler decisions, adding or removing ECS instances as the workload evolves.

# 4 JOINT SCHEDULING FOR HETEROGENEOUS RESOURCES

A central lesson from over-subscription in production is that perfect prediction is not the real bottleneck. Even if a scheduler produces a very accurate estimate $\widetilde{U}_i(t)$, small forecast errors remain unavoidable under workload shifts and bursts, and these errors are amplified at high utilization, causing constraint violations, OOM events, and SLO degradation.

We argue that the root cause is not prediction accuracy but the **rigid per-machine memory boundary**: when every machine has a fixed, impermeable memory ceiling, even a small spike that overshoots capacity causes an OOM failure rather than a recoverable overload. Prior work attacks this by improving prediction quality, but a 100% correct forecast is fundamentally unattainable in production, and prediction errors are amplified into costly failures at high utilization. Remote memory breaks this rigidity by providing an elastic overflow path: when a spike exceeds local capacity, excess demand spills to the remote pool at microsecond latency instead of crashing. This transforms memory management from a *hard failure* problem into a *resource allocation* problem—how much remote capacity to provision, and how to schedule it across tenants—which our joint scheduler formalizes.

## 4.1 Problem Definition

We consider a cluster with a set of compute nodes (machines) $\mathcal{M} = \{M_1, M_2, ..., M_m\}$. Each machine $M_j$ has a local memory capacity $C_j^{\mathrm{L}}$ and provides CPU resources for query execution. We schedule a set of tenants/instances $\mathcal{I} = \{I_1, I_2, ..., I_n\}$. Each instance $I_i$ runs one or more workers; we denote the set of all workers by $\mathcal{W}$ and write $w \in \mathcal{W}$.

Let $\widehat{U}_w(t)$ denote the (unknown) future memory demand time series of worker $w$. The scheduler assigns a local quota $q_w^{\mathrm{L}}$. At time $t$, if the demand exceeds the local quota, the worker borrows the fall-short from remote memory:

$$B_w(t) = \max\{0,\, \widehat{U}_w(t) - q_w^{\mathrm{L}}\}. \tag{1}$$

Since remote memory is assumed elastically provisioned, there is no upper bound on $B_w(t)$ at the system level; instead, remote borrowing is controlled by an SLO-style constraint below.

*4.1.1 Local bin packing constraint (local DRAM packing).* Let $\phi : \mathcal{W} \to \mathcal{M}$ be the placement function that maps each worker to a compute node. For each machine $M_j$, let $\mathcal{W}(M_j) = \{w \in \mathcal{W} \mid \phi(w) = M_j\}$ be the workers placed on it. We require local quotas to fit within each machine's local capacity:

$$\sum_{w \in \mathcal{W}(M_j)} q_w^{\mathrm{L}} \le C_j^{\mathrm{L}}, \quad \forall M_j \in \mathcal{M}. \tag{2}$$

*4.1.2 Remote-borrowing ratio constraint.* Although remote capacity is elastic, remote memory has a performance penalty compared with local DRAM and consumes network bandwidth. Thus, remote borrowing should account for only a small fraction of each worker's memory footprint. We formalize this with the *remote-borrowing ratio*, defined as the fraction of a worker's aggregate memory consumption that is served from the remote pool over a horizon $\mathcal{T}$:

$$\rho_w = \frac{\sum_{t \in \mathcal{T}} B_w(t)}{\sum_{t \in \mathcal{T}} \widehat{U}_w(t)}. \tag{3}$$

We enforce a per-worker bound:

$$\rho_w \le \epsilon_w, \quad \forall w \in \mathcal{W}, \tag{4}$$

where $\epsilon_w$ is a policy parameter (e.g., $\epsilon_w = 2\%$), capping each worker's time-averaged remote footprint relative to its total memory usage.

*4.1.3 Objective (TCO minimization).* We optimize a single Total Cost of Ownership (TCO) objective over compute machines and remote memory. Because remote memory is assumed elastically provisioned, its cost is incurred by the *provisioned* remote capacity required to sustain peak borrowing. Let the provisioned remote capacity be

$$C^{\mathrm{R}} \triangleq \max_{t \in \mathcal{T}} \sum_{w \in \mathcal{W}} B_w(t), \tag{5}$$

i.e., the maximum aggregate remote borrowing over the horizon. **Remote capacity constraint.** In production, the PACE cluster has finite capacity $C_{\max}^{\mathrm{R}}$. We add the constraint:

$$C^{\mathrm{R}} \le C_{\max}^{\mathrm{R}}. \tag{6}$$

In AnalyticDB's deployment, PACE is provisioned with 3–5× the peak observed aggregate borrowing, ensuring this constraint rarely binds. **Separate cost coefficients.** Compute and remote memory have different unit prices. Let $\alpha$ denote the normalized cost per compute machine and $\beta$ the cost per unit of remote memory. In production, $\alpha/\beta \approx 2$–$3$ (compute is more expensive). The revised objective is:

$$\min_{\phi,\, \{q_w^{\mathrm{L}}\}} \; \alpha \cdot |\{M_j \in \mathcal{M} \mid \mathcal{W}(M_j) \neq \emptyset\}| \; + \; \beta \cdot C^{\mathrm{R}} \quad \text{s.t. } (2), (4), (6) \tag{7}$$

This objective captures the intended trade-off: tighter packing reduces the number of machines but may increase remote borrowing and thus $C^{\mathrm{R}}$, while larger local quotas reduce remote borrowing at the cost of additional machines.

## 4.2 Scheduling Algorithm

We present a practical scheduling framework to approximately solve Eq. (7). The key is to translate the borrowing-ratio constraint into a quota rule using forecasts: we choose $q_w^{\mathrm{L}}$ as a high quantile of the predicted demand, which upper-bounds the expected remote-borrowing ratio $\rho_w$ (since $\rho_w \le \Pr[\widehat{U}_w > q_w^{\mathrm{L}}]$ when excess-over-quota is bounded by a constant factor of $\widehat{U}_w$), then pack workers based on $q_w^{\mathrm{L}}$, and finally estimate the implied remote capacity $C^{\mathrm{R}}$. The framework includes an online algorithm and an offline algorithm (periodic repacking).

**Algorithm 1** Online Bin Packing for Local DRAM

**Require:** Machines $\mathcal{M}$ with local capacities $\{C_j^{\mathrm{L}}\}$ and current residual capacities $\{\mathrm{Rem}_j^{\mathrm{L}}\}$.
**Require:** Arriving workers $\mathcal{W}_{\mathrm{in}}$; local reservation requests $\{r_w\}_{w\in\mathcal{W}_{\mathrm{in}}}$.
**Ensure:** Placement $\phi(w)$ and local quotas $\{q_w^{\mathrm{L}}\}$ for $w \in \mathcal{W}_{\mathrm{in}}$.

**for all** $w \in \mathcal{W}_{\mathrm{in}}$ **do**
  $q_w^{\mathrm{L}} \leftarrow r_w$ ▷ no prediction; use declared reservation
**end for**
Sort $\mathcal{W}_{\mathrm{in}}$ by descending $q_w^{\mathrm{L}}$ ▷ optional, if a batch arrives
**for all** $w \in \mathcal{W}_{\mathrm{in}}$ **do**
  $\mathcal{F} \leftarrow \{M_j \in \mathcal{M} \mid \mathrm{Rem}_j^{\mathrm{L}} \geq q_w^{\mathrm{L}}\}$ ▷ feasible machines
  **if** $\mathcal{F} = \emptyset$ **then**
    **scale-out** compute (open a new machine) and update $\mathcal{M}, \mathrm{Rem}^{\mathrm{L}}$
    Recompute $\mathcal{F}$
  **end if**
  $M^\star \leftarrow \arg\min_{M_j\in\mathcal{F}} \left(\mathrm{Rem}_j^{\mathrm{L}} - q_w^{\mathrm{L}}\right)$ ▷ best-fit
  $\phi(w) \leftarrow M^\star$
  $\mathrm{Rem}_{M^\star}^{\mathrm{L}} \leftarrow \mathrm{Rem}_{M^\star}^{\mathrm{L}} - q_w^{\mathrm{L}}$
**end for**

*4.2.1 Online Scheduling.* The online scheduler handles incremental events (instance creation, scale-out, and worker restart) and must make placement decisions without future knowledge. In particular, we treat memory demand as *unknown* at admission time; hence the online algorithm is an *online bin packing* procedure that uses only (i) machine capacities and (ii) the worker's declared local reservation request (or a conservative default), rather than any forecast $\widetilde{U}_w(t)$. The online output is: (1) a placement decision $\phi(w)$ and (2) a local quota $q_w^{\mathrm{L}}$. Remote memory is borrowed on demand according to (1); the system provisions remote capacity offline from observed peak demand.

When a worker $w$ arrives, it specifies (or the system assigns) a local reservation $r_w$ (bytes), which is interpreted as the amount of local DRAM the worker must always have available. The online scheduler sets $q_w^{\mathrm{L}} \leftarrow r_w$, and performs placement using an online bin packing heuristic. The remote-borrowing ratio constraint (4) is *not* enforced online (since $\rho_w$ depends on unknown future demand); instead, it is monitored and enforced by the offline controller via quota tuning and/or migrations (Section 4.2.2).

We use Best-Fit (or First-Fit) decreasing over the current arrival batch when multiple workers arrive together; for purely sequential arrivals, Best-Fit reduces to choosing the feasible machine with the smallest remaining local capacity after placement.

Algorithm 1 sets each worker's local quota to its reservation request, sorts batched arrivals by decreasing quota, and applies best-fit placement over feasible machines, triggering scale-out when none is feasible.

*4.2.2 Offline Scheduling.* The offline scheduler runs periodically to combat **consolidation drift**—fragmentation caused by worker arrivals, departures, and quota changes over time—and to re-estimate the required remote capacity. Migrations follow a **drain-then-migrate** protocol: workers are relocated only during query-idle periods after active queries drain, with no in-flight query disruption. Only cache metadata (KB-scale) is transferred; remote data blocks remain in place on PACE and are re-attached via RDMA without relocation, limiting per-worker downtime to ∼2–5 seconds, with

**Algorithm 2** Offline Repacking and Remote Capacity Estimation

**Require:** Current placement $\phi$; machines $\mathcal{M}$ with capacities $\{C_j^{\mathrm{L}}\}$; migration budget $B$.
**Require:** Workers $\mathcal{W}$; updated predicted series $\{\widetilde{U}_w(t)\}_{w\in\mathcal{W}}$ over horizon $\mathcal{T}$; borrowing-ratio bounds $\{\epsilon_w\}$.
**Ensure:** New placement $\phi_{\mathrm{new}}$, local quotas $\{q_w^{\mathrm{L}}\}$, and estimated remote capacity $\widetilde{C}^{\mathrm{R}}$.

**for all** $w \in \mathcal{W}$ **do**
  $q_w^{\mathrm{L}} \leftarrow \mathrm{P}_{(1-\epsilon_w)\text{-th}}(\widetilde{U}_w(t))$
**end for**
Sort $\mathcal{W}$ by descending $q_w^{\mathrm{L}}$
$\phi_{\mathrm{new}} \leftarrow \emptyset$
**for all** $w \in \mathcal{W}$ **do**
  Choose $M^\star \leftarrow$ BestFitLocal$(w, \mathcal{M})$ under $\phi_{\mathrm{new}}$
  **if** local capacity fits on $M^\star$ **then**
    $\phi_{\mathrm{new}}(w) \leftarrow M^\star$
  **else**
    open a new machine
  **end if**
**end for**
$\mathcal{K} \leftarrow \{w \in \mathcal{W} \mid \phi_{\mathrm{new}}(w) \neq \phi(w)\}$ ▷ migration set
**if** $|\mathcal{K}| > B$ **then**
  Keep top-$B$ workers in $\mathcal{K}$ with largest consolidation gain; revert others to $\phi$
**end if**
Estimate $\widetilde{C}^{\mathrm{R}} \leftarrow \max_{t\in\mathcal{T}} \sum_{w\in\mathcal{W}} \max\{0, \widetilde{U}_w(t) - q_w^{\mathrm{L}}\}$ ▷ peak borrowing
Apply $\phi_{\mathrm{new}}$ with bounded-rate migrations

the migration budget capped at $B = 10$–$50$ workers per scheduling round to bound aggregate migration traffic. Concretely, at $B = 50$ workers × KB-scale metadata per worker, the aggregate migration payload per round is on the order of a few MB, well below the 100 Gbps ToR capacity even when compressed into a single burst; the migration window is further scheduled at off-peak hours and isolated per-tenant, so ToR saturation is not a concern in production. The scheduler compares updated predictions against the prior placement to recompute local quotas, repacks the cluster in batch under the migration budget $B$, and estimates the implied remote capacity from the predicted aggregate borrowing peak.

Algorithm 2 sets each worker's local quota to the $(1-\epsilon_w)$-th percentile of its updated predicted usage, which upper-bounds the remote-borrowing ratio $\rho_w$ (Eq. 3) by $\epsilon_w$ under bounded-excess distributions, and then sorts workers by decreasing quota for best-fit repacking. When the candidate migration set exceeds the budget $B$, workers are ranked by *consolidation gain*—the number of machines a move would free—and only the top-$B$ moves are committed, preferring relocations that vacate a machine entirely over marginal rebalancing. The implied remote capacity $\widetilde{C}^{\mathrm{R}}$ is sized to the predicted aggregate peak borrowing $\max_{t\in\mathcal{T}} \sum_w \max\{0, \widetilde{U}_w(t) - q_w^{\mathrm{L}}\}$ over the horizon, and feeds directly into the TCO objective Eq. (7).

Unlike local-only schedulers whose forecast errors translate directly into OOM at high utilization, a mis-sized $q_w^{\mathrm{L}}$ here does *not* cause OOM: any excess demand flows through Eq. (1) to remote memory at microsecond latency. Under-estimation merely raises the realized borrowing ratio $\rho_w$ toward the bound $\epsilon_w$ and is corrected at the next offline round; over-estimation wastes a small amount of local DRAM. The offline scheduler therefore relies on

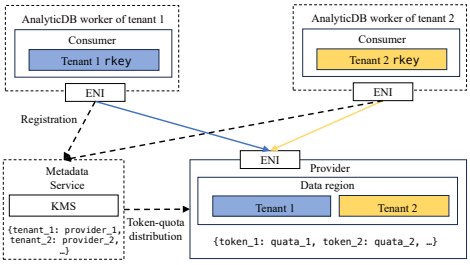



**Figure 3: Architecture of PACE.**

predictions for *optimization*, not for *safety*—a property that decouples our scheduling correctness from the fundamental limits of demand forecasting.

# 5 PACE: MULTI-TENANT SECURE REMOTE MEMORY SYSTEM

Existing RDMA-based remote memory systems lack hardware-enforced tenant isolation, as discussed in Section 2.2. An alternative design partitions a single shared pool by application-level encryption, which eliminates per-tenant fragmentation but adds 0.3–0.5 $\mu$s of crypto overhead per block (10–25% of the 2–5 $\mu$s RDMA latency) and consumes CPU cycles on every access, undermining the CPU-bypass property of one-sided RDMA that Narwhal's cache path depends on. PACE instead uses **commodity RDMA NICs**—no custom hardware is required. Protection Domain isolation is a standard RDMA feature available on all commodity RNICs: each tenant is assigned a distinct PD, and hardware-level PD checks at the NIC run at line rate with zero CPU cycles and no crypto overhead. Per-tenant fragmentation—the primary drawback of hardware separation—is manageable through periodic offline repacking, generous headroom (3–5× peak borrowing), and moderate per-node tenant counts (2–5 in production). PACE is, to our knowledge, the first RDMA shared memory pool designed for multi-tenant analytical databases that enforces security and performance isolation through commodity NIC hardware. This section presents the architecture, isolation model, resource management, and fault-tolerance design of PACE.

## 5.1 Architecture Overview

Figure 3 illustrates the PACE architecture, which comprises three layers: a *consumer side*, a *provider side*, and a centralized *metadata service*.

Each AnalyticDB worker acts as a *consumer* of remote memory. Upon creation, the worker registers with the metadata service and receives a tenant-specific `rkey` along with one or more RDMA Queue Pairs (QPs) bound to its own Protection Domain (PD). All remote memory operations issued by the instance—cache evictions via `RDMA WRITE`, cache lookups via `RDMA READ`, and allocation/free requests—carry the tenant's `rkey` and are confined to the associated PD. Each consumer connects to provider nodes through a user-space Elastic Network Interface (ENI), which provides network-level isolation between tenants at the transport layer.

Each *provider* node registers a portion of its local DRAM with the RDMA NIC and exposes it as part of the global memory pool. The registered memory is partitioned into per-tenant *data regions*, each backed by a dedicated Memory Region (MR) with a unique `rkey`. A single provider node may host data regions for multiple tenants (e.g., User1 and User2 in Figure 3), but each region is bound to a distinct PD, ensuring hardware-level separation. Providers also maintain *token-to-quota* mappings that record the capacity allocated to each tenant on that node. These mappings are distributed by the metadata service and enforced locally by the provider.

The *metadata service* is logically centralized and physically replicated to coordinate the control plane. It performs four functions: (1) node registration, where provider nodes register their available capacity and network endpoints upon startup; (2) tenant-to-provider mapping, maintaining which provider nodes hold data regions for each tenant and distributing this mapping to consumers; (3) token and quota distribution, issuing per-tenant capacity quotas as tokens to providers, which enforce them locally; and (4) address-mapping updates, propagating updated remote addresses and `rkey` credentials to affected consumers when data regions are created, migrated, or reclaimed.

## 5.2 Hardware-Enforced Tenant Isolation

The central security mechanism of PACE is the mapping of each tenant to a distinct RDMA Protection Domain. A tenant's QPs, MRs, and CQs are all bound to its PD. When a consumer issues an RDMA request to a provider, the NIC performs the following verification at line rate, entirely in hardware:

$$\textsc{Allow} \iff (\text{req.PD} = \text{MR.PD}) \wedge (\text{op} \in \text{MR.permissions}). \quad (8)$$

If the requesting QP's Protection Domain does not match the target MR's Protection Domain, or if the requested operation (e.g., `WRITE` on a read-only region) is not permitted, the NIC silently drops the packet and posts a completion error. No data is exposed to the requester.

This mechanism provides three properties. *First*, it guarantees *cross-tenant security*: even if a malicious tenant somehow obtains another tenant's `rkey`, the cross-PD check rejects the access at the NIC, before any data leaves the provider's memory bus. *Second*, it incurs *zero software overhead*: the verification is performed in the NIC's hardware pipeline and adds no measurable latency to the RDMA data path, with no OS kernel involvement. *Third*, it supports *intra-instance isolation*: the same PD mechanism extends naturally to workers within a single database instance—when fine-grained isolation is required (e.g., different workload groups sharing an instance), each worker group can be assigned its own PD, preventing one group's RDMA operations from interfering with another's memory regions.

All MR keys (`rkey`) are issued as short-lived ephemeral credentials. A trusted key management service (KMS) periodically rotates keys and distributes fresh tokens to authorized consumers via the metadata service. Key rotation bounds the window of exposure in case of credential leakage. Upon rotation, old `rkey` values are invalidated at the provider by re-registering the MR with a new key; any in-flight RDMA request carrying a stale `rkey` fails with a remote-access error and is retried by the consumer with the refreshed credential.

## 5.3 Resource Allocation and QoS Enforcement

Hardware-enforced PD isolation prevents unauthorized access but does not, by itself, prevent resource starvation: a single tenant issuing a burst of RDMA requests can monopolize provider bandwidth and starve co-located tenants. PACE therefore layers QoS mechanisms on top of PD-based isolation.

**Memory quotas.** Each tenant's data region on a provider is bounded by a configurable capacity limit, the *quota* shown in Figure 3. When a tenant's allocation request would cause its aggregate remote footprint to exceed its quota, the provider returns a graceful failure code, and the consumer falls back to evicting within its remote cache region or serving the miss from disk. Quotas are dynamically adjusted by the Joint Resource Scheduler (Section 4) based on predicted demand and SLO priority, allowing the system to redistribute remote capacity as workload profiles shift.

**Weighted fair queuing (WFQ).** RDMA work requests arriving at provider-side receive queues are scheduled according to per-tenant SLA weights, ensuring that latency-sensitive workloads are not starved by bulk scan traffic from lower-priority tenants. However, Kong et al. [17] show that on commodity RNICs, WFQ and bandwidth throttling cannot fully protect shared on-NIC microarchitectural resources (QP context caches, translation tables, pipelines) from noisy-neighbor contention. We scope our claims accordingly: PD provides **full security isolation**, while performance isolation is **partial**. Kong et al.'s attack requires an *untrusted* tenant that can issue arbitrary RDMA verbs; this premise does not hold in Narwhal, where tenants never call RDMA directly—all accesses go through AnalyticDB's storage kernel (part of the TCB), which exposes only a fixed set of data-path verbs, enforces per-tenant verb rates and WR caps, and confines control-path verbs to the operator management plane. SR-IOV VFs and RDMA traffic classes provide additional coarse-grained hardware partitioning. The 1.63% remote-borrowing ratio *measured in §7.1* and type-aware placement are supporting (not load-bearing) mitigations; a dedicated indirection layer remains future work.

## 5.4 Fault Tolerance

PACE addresses the independent failure modes of disaggregation with three mechanisms. Tenant-to-region mappings, quotas, and `rkey` bindings are persisted in a Raft-based coordination service tolerating minority replica failures. On tenant deletion, data regions are marked *expired* and reclaimed asynchronously by a background collector, remaining PD-protected until freed to prevent stale-data exposure. On consumer failure and restart, the recovered process re-registers with the metadata service, obtains fresh `rkey`/PD credentials, and re-attaches to its intact data regions on surviving providers—no data migration, minimal downtime, and no cache-repopulation I/O amplification.

# 6 MEMORY ALLOCATOR

The Memory Allocator manages data placement across a three-tier hierarchy: local DRAM (L1), RDMA-accessible remote memory (L2), and disk (L3). The block cache is logically partitioned into a *local cache region* (L1) and a *remote cache region* (L2); both are indexed by the same hash table and share a unified lookup interface, but reside on different physical tiers.

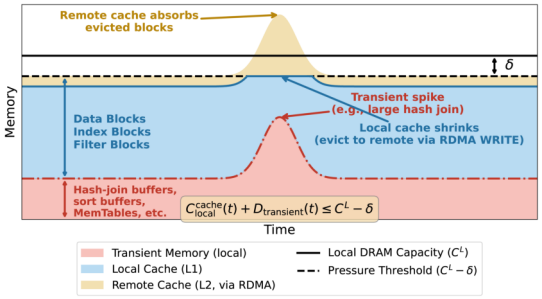



**Figure 4: Dynamic local/remote capacity adjustment in the Memory Allocator.**

This section presents the two core mechanisms of the Memory Allocator: dynamic local/remote capacity adjustment for OOM avoidance (Section 6.1), and type-aware data placement that controls the distribution of different block types across the two cache tiers (Section 6.2).

## 6.1 Dynamic Local/Remote Capacity Adjustment

In production, the memory consumed by an AnalyticDB worker falls into two categories (illustrated in Figure 4): *cache-managed memory* (block cache, bounded and controllable) and *transiently allocated memory* (hash-join buffers, sort buffers, MemTables, network buffers, etc., which are demand-driven and bursty). When a transient spike occurs, the block cache still holds its budgeted capacity, leaving insufficient local DRAM and causing OOM. Shrinking the cache proactively wastes capacity during normal operation; shrinking it reactively may be too slow.

The Memory Allocator resolves this tension by dynamically migrating cached blocks between local and remote memory based on real-time pressure, as shown by the interplay between the blue (local cache) and yellow (remote cache) regions in Figure 4.

*6.1.1 Pressure-Threshold Invariant.* Let $C^{\mathrm{L}}$ denote the total local DRAM capacity (solid horizontal line in Figure 4), $C^{\mathrm{cache}}_{\mathrm{local}}(t)$ the local cache capacity at time $t$, and $D_{\mathrm{transient}}(t)$ the current transient memory demand. The Memory Allocator enforces:

$$C^{\mathrm{cache}}_{\mathrm{local}}(t) + D_{\mathrm{transient}}(t) \le C^{\mathrm{L}} - \delta, \tag{9}$$

where $\delta$ is a configurable safety headroom (the gap between the solid and dashed horizontal lines in Figure 4). This invariant guarantees that local DRAM is never exhausted: even under a sudden transient spike, a $\delta$-sized buffer remains available for OS-level allocations, thread stacks, and other unmanaged memory.

The safety headroom $\delta$ is determined by analyzing the historical distribution of transient memory demand $D_{\mathrm{transient}}(t)$ over a representative observation window. A more conservative $\delta$ provides a wider safety margin at the cost of reduced local cache capacity; a tighter $\delta$ increases local cache headroom but narrows the buffer. The appropriate tradeoff depends on workload volatility and is tuned per cluster based on production monitoring data.

During steady state (the left and right portions of Figure 4), transient demand is low and the local cache occupies a large share of DRAM, with the sum $C^{\mathrm{cache}}_{\mathrm{local}}(t) + D_{\mathrm{transient}}(t)$ remaining well below $C^{L} - \delta$. When a transient spike occurs (the center of the

figure), $D_{\text{transient}}(t)$ rises sharply, and the allocator must reduce $C_{\text{local}}^{\text{cache}}(t)$ to maintain the invariant.

*6.1.2 Eviction and Promotion Protocol.* When $D_{\text{transient}}(t)$ rises and threatens to violate (9), the allocator proceeds through four phases, corresponding to the transition visible in the center of Figure 4:

(1) **Pressure detection.** The allocator periodically samples OS allocator statistics (e.g., `jemalloc` active bytes) and process RSS. A pressure event fires when $C_{\text{local}}^{\text{cache}}(t) + D_{\text{transient}}(t) > C^{\text{L}} - \delta$.
(2) **Local-to-remote eviction.** Victim blocks are selected from the local cache via the standard replacement policy (LRU/Clock) and written to the remote memory pool via one-sided RDMA `WRITE`. The cache index is updated in place: each evicted entry records its remote address and RDMA `rkey`. In the figure, this phase corresponds to the shrinkage of the blue region and the simultaneous growth of the yellow region.
(3) **Remote-tier serving.** Subsequent lookups for evicted blocks are served via RDMA `READ` at microsecond-scale latency, rather than falling through to millisecond-scale disk I/O. The unified hash-table index transparently redirects lookups to the remote tier, so upper layers of the query engine are unaware of the tier change.
(4) **Promotion on pressure relief.** When $D_{\text{transient}}(t)$ drops (the right-hand recovery in Figure 4), frequently accessed remote-resident blocks are promoted back to local DRAM via RDMA `READ`, restoring full local-memory performance. The local cache expands and the remote cache contracts, returning to the steady-state configuration.

If a tenant's remote quota is exhausted (not yet observed in production, since PACE auto-scales), the Memory Allocator spills victims directly to local SSD and marks them disk-resident; quotas sized at 3–5× peak borrowing and the borrowing-ratio bound (Eq. (4)) keep this path rare.

Compared with naïve alternatives—proactively shrinking the cache or evicting to disk—this mechanism provides three advantages: (1) *elastic headroom*: the PACE pool auto-scales with demand, so remote capacity is provisioned generously and has never been exhausted in production; (2) *graceful degradation with cheaper eviction*: compared with **reactive cache shrinking**, which must synchronously evict entries, tear down refcounts, and invoke allocator-level reclamation on the critical path (tens of $\mu$s of work that further contends with query threads on the same cores), an RDMA `WRITE` to remote memory completes in 2–5 $\mu$s on a one-sided, CPU-bypass path—strictly cheaper on the allocation-pressure path. Spilling to local SSD costs ~100 $\mu$s per block and drops the spilled data out of the microsecond tier. Evicting to remote memory is therefore both the fastest eviction target and the one that keeps spilled blocks addressable at microsecond latency rather than dropping them to the disk tier—cache effectiveness degrades smoothly instead of collapsing; (3) *enlarged effective cache*: the total cache ($C_{\text{local}}^{\text{cache}} + C_{\text{remote}}^{\text{cache}}$) exceeds local DRAM, benefiting steady-state performance without wasting local memory on idle headroom. As visible in Figure 4, the combined blue and yellow regions always exceed the local DRAM capacity line, meaning the system maintains a larger effective cache than what local memory alone could provide.

## 6.2 Optimized Data Placement across Local and Remote Cache

Section 6.1 determines *how much* data moves between tiers; this subsection determines *which* data should preferentially stay local and which can tolerate remote placement.

The block cache holds three types of blocks from SST files: `Data Blocks`, `Index Blocks`, and `Filter Blocks` (See Section 2.3 for more details).

`Index` and `Filter` blocks gate the lookup path: a miss on either triggers additional I/O *before* the target data can be read. A few extra microseconds of remote-memory latency on a `Data Block` is far less costly than the cascading I/O caused by a metadata miss. Yet under a unified eviction policy, bulky `Data Blocks` dominate the cache footprint and evict metadata, causing *metadata thrashing*—a sharp rise in random I/O per lookup and a sudden performance cliff.

We enforce a maximum fraction of the local cache that `Data Blocks` may occupy, controlled by a parameter `data_block_max_ratio` $r \in (0, 1)$. Let $C$ be the total local primary cache capacity: $C_{\text{data}} \leq r \cdot C$, where $C_{\text{data}}$ is the current local cache footprint of `Data Blocks`. The remaining budget $(1-r) \cdot C$ is reserved for `Index/Filter` blocks and other metadata.

- `Index/Filter` blocks are admitted to the local cache with higher priority. They are evicted only when the metadata region itself is full, following the standard replacement policy within that region.
- When a `Data Block` insertion would violate $C_{\text{data}} \leq r \cdot C$, eviction is performed *within the data-block region only*; metadata blocks are never evicted to accommodate data blocks.
- Data blocks evicted under the cap are placed in the *remote cache region*, remaining accessible at RDMA latency rather than being dropped to disk.

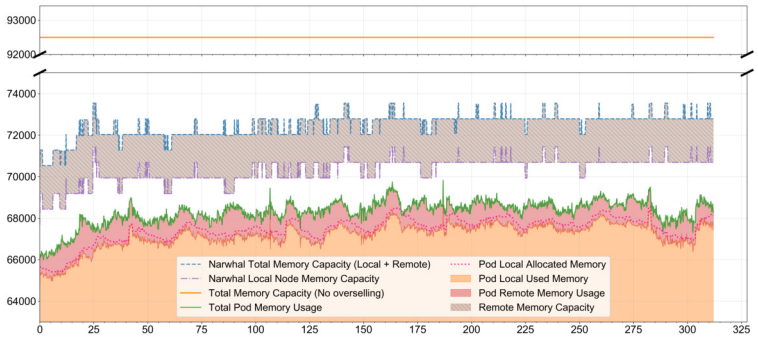
This layered design ensures that the Memory Allocator makes *database-aware* decisions: the engine's knowledge of block types and their latency sensitivity directly governs which data resides in fast local DRAM, which in elastic remote memory, and which falls through to disk.

# 7 EVALUATION

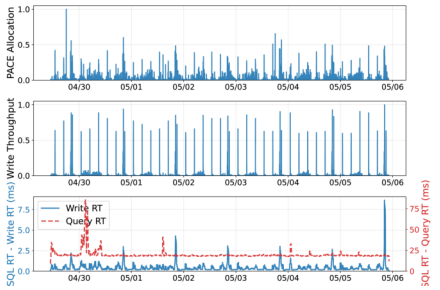
## 7.1 End-to-end Evaluation

To evaluate the integrated system in a production setting, we deployed Narwhal on an AnalyticDB cluster and collected metrics over a 7-day window. Figure 5(a) shows the dynamic interaction between local and remote memory. When a pod's local memory is insufficient, it borrows from the remote pool via RDMA; the scheduler then reallocates local capacity to restore normal operation. Narwhal raises the memory overselling rate from 92.67% to 121.89% and node utilization from 78.16% to 96.63%, with the average remote-borrowing ratio per pod at 1.63% of total memory usage (max 2.90%), keeping performance degradation negligible.

Figure 5(b) reports PACE-side metrics from a production instance that was deliberately selected for its heavy remote memory usage. A clear diurnal pattern is visible: write throughput peaks during nightly batch ingestion periods each day, and PACE remote memory allocation rises correspondingly to absorb the transient write




(a) Local and remote memory capacity and usage over time.




(b) PACE allocation, write throughput, write SQL mean RT, and query RT over a 7-day production window.

**Figure 5: End-to-end production evaluation.**

pressure. Despite these periodic load surges, write SQL mean response time remains stable in the single-digit millisecond range, and query response time stays within tens of milliseconds. This confirms that PACE sustains low-latency remote memory access under real production workload dynamics without introducing noticeable performance jitter.

## 7.2 Schedule

We conducted experiments using pod runtime data from a specific region of the AnalyticDB production environment. In the experiments, we used 14 days of runtime data from 800+ pods in a specific region of the online environment to simulate our algorithm and compared it with four overselling algorithms: no memory overselling, fixed 120% overselling rate, sliding window, and Eigen+ [25].

We compared our algorithm with four other overselling algorithms: no memory overselling (*BASE*), fixed 120% overselling rate (*FIXED*), sliding window (*SW*), which is representative of prediction-driven approaches widely adopted in production systems such as Google Autopilot[34], and *Eigen+* [25], the closest prior work on database memory oversubscription. As shown in Figure 6, compared to SW, our algorithm improves single-node memory overselling by up to 21% and the average by 4%; although SW's overselling rate is close to ours, up to 10 nodes experience OOM for 6% of the total time, and FIXED encounters OOM almost continuously. Against Eigen+—which treats each machine as a closed bin with a hard memory ceiling—Narwhal achieves a 15% higher oversubscription ratio while requiring remote capacity of only ~half the local DRAM footprint and keeping the realized remote-borrowing ratio within the policy bound $\epsilon_w = 2\%$ (consistent with the 1.63% independently measured in §7.1), transforming hard OOM failures into soft overload absorbed by remote memory.

## 7.3 Multi-tenant

We evaluate Narwhal's multi-tenant performance and security across four dimensions. For latency and bandwidth, we compare Narwhal, PACE, and cloud disk under varying client counts and concurrency levels. As shown in Figure 7a, Narwhal's read/write latency is 47%–76% lower than cloud disk, and Figure 7b shows corresponding bandwidth improvements. For multi-tenant overhead, as shown in Figure 7c, Narwhal's latency is essentially on par with PACE under various client numbers and concurrency levels, and in some scenarios, it performs even better than PACE, indicating that the multi-tenancy feature has a negligible impact on latency. We also tested latency variation with the number of clients, shown in Figure 7d. As the number of clients increases, Narwhal's average latency does not increase significantly, showing that PD isolation, WFQ, and bandwidth throttling mitigate bandwidth-related noisy-neighbor interference; on-NIC microarchitectural contention [17] is only partially addressed at the traffic-shaping layer, but the closed, operator-controlled storage-kernel verb path (Section 5.3) is the primary mitigation that prevents the adversarial premise from applying, with the 1.63% remote-borrowing ratio *measured in §7.1* serving as a supporting (not load-bearing) margin. For isolation under skewed tenant load, we measure small-request latency while competing tenants generate skewed elephant flows at increasing aggregate bandwidth up to 100 Gbps NIC saturation. As shown in Figure 7e, latency remains essentially unchanged across the entire pressure range (Avg CV = 1.89%), attributable to QP-level hardware isolation and HPCC-RTT congestion control preventing queue buildup.

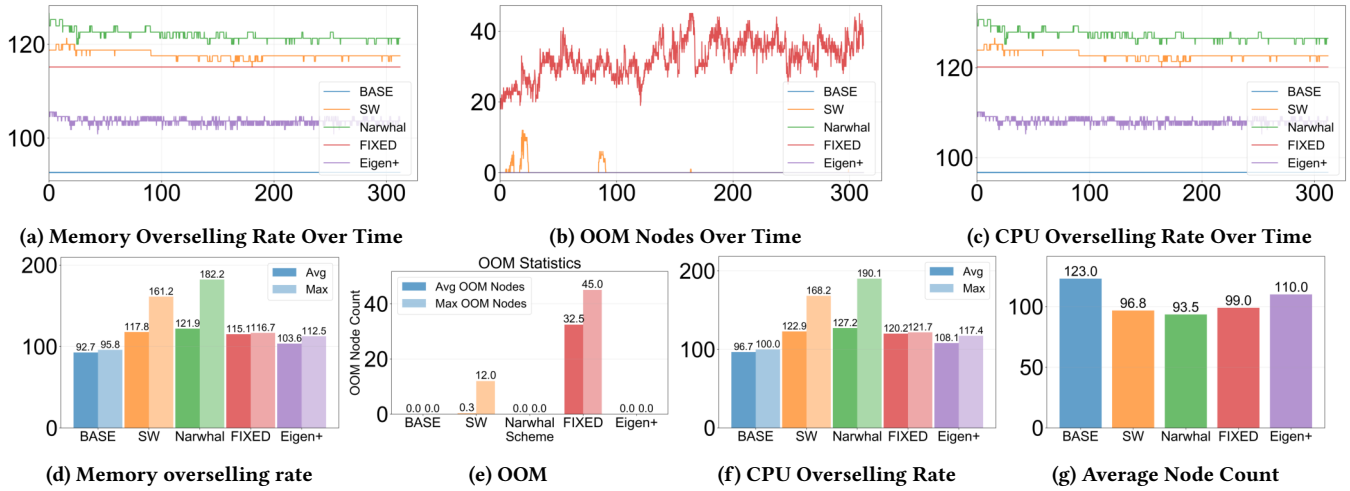
For security, we issue 100,000 cross-access attempts by 32 clients bound 1-to-1 to 32 tenants (90,000 unauthorized, 10,000 legitimate); all unauthorized accesses are rejected by PACE and all legitimate ones succeed.

## 7.4 Performance

We present two micro-benchmarks validating Narwhal's core performance.

*7.4.1 Cache-Size Sensitivity Under Skewed Access.* We load a TPC-H `lineitem` subset into AnalyticDB, using all columns as the composite primary key to exercise the full index–filter–data lookup path, with record access probabilities drawn from an exponential distribution to induce pronounced temporal locality. We sweep the local block cache capacity from $400 \times 10^3$ (full size) down to near zero, and measure cache hit ratio, mean/median query latency ($\mu$s), p75, and p99.9 tail.

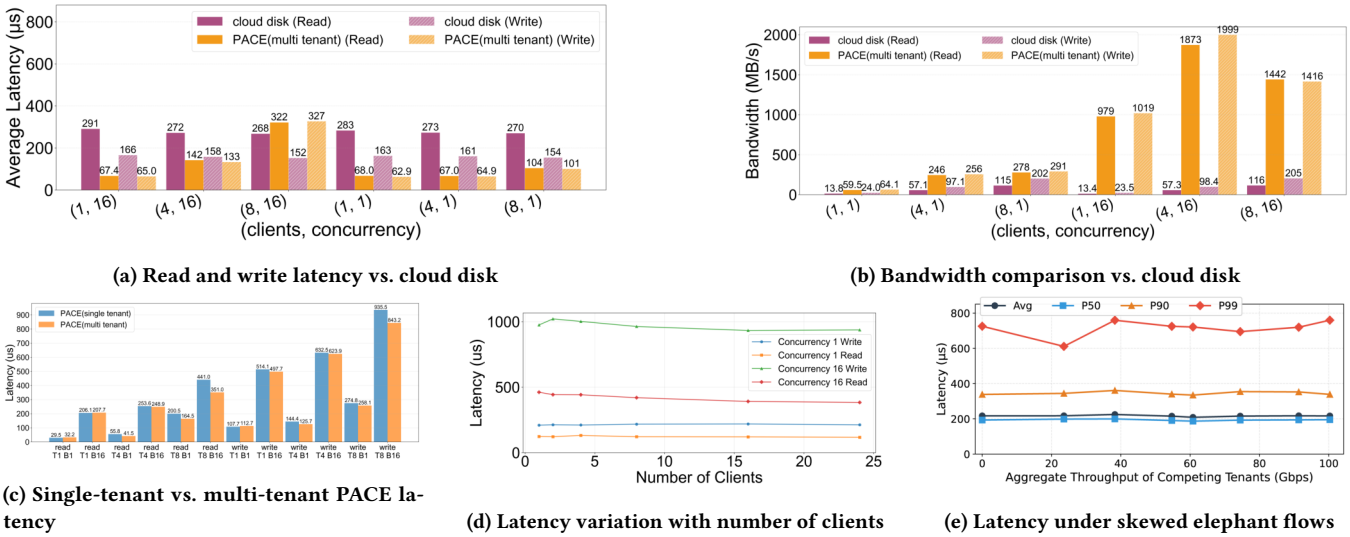
Figure 8 shows three regimes as cache shrinks: a *stable* region ($\geq 200 \times 10^3$, hit ratio $\geq$0.9, flat latencies), a *graceful degradation* region ($50$–$200 \times 10^3$, hit ratio declining $0.9 \rightarrow 0.4$, latency up <30%), and a *collapse* region ($< 50 \times 10^3$, hit ratio near zero, tail latencies spike). Thus a substantial fraction of local cache can be offloaded to the remote tier without crossing into collapse, provided




(a) Memory Overselling Rate Over Time (b) OOM Nodes Over Time (c) CPU Overselling Rate Over Time

(d) Memory overselling rate (e) OOM (f) CPU Overselling Rate (g) Average Node Count

**Figure 6: Comparison of five algorithms**




(a) Read and write latency vs. cloud disk

(b) Bandwidth comparison vs. cloud disk

(c) Single-tenant vs. multi-tenant PACE latency

(d) Latency variation with number of clients

(e) Latency under skewed elephant flows

**Figure 7: Performance comparison of single-tenant PACE, multi-tenant PACE, and cloud disk.**

access locality is present. Real AnalyticDB workloads exhibit even stronger locality than the synthetic exponential model (index/filter blocks form 1:N hot spots; partition-aligned ingestion concentrates data-block accesses), so this experiment is a conservative lower bound on the locality headroom.

*7.4.2 Type-Aware Placement and the Data-Block Cap.* We use the same TPC-H `lineitem` ingestion but draw queries uniformly to eliminate locality. We vary the *data-block cap ratio* $r \in \{0.1, 0.55, 1.0\}$—the maximum fraction of L1 cache that `Data Blocks` may occupy (Section 6.2; $r = 1.0$ is the type-oblivious baseline)—and sweep total L1 size from $50 \times 10^3$ to $400 \times 10^3$.

Figure 9 presents the six metrics; three observations follow.

**Index hit rate and the performance cliff.** With $r = 1.0$ (no cap, bottom-left panel), index-block hit ratio saturates at ~65% even at the largest cache size because bulky `Data Blocks` continuously evict index entries. Setting $r = 0.1$ drives index hit ratio to 100% already at $\approx 150 \times 10^3$, roughly 2× earlier than $r = 0.55$—the *metadata thrashing* pathology of Section 6.2.

**Throughput and latency.** At small-to-medium cache sizes ($< 200 \times 10^3$), $r = 0.1$ sustains ~10–20k more ops/s than $r = 1.0$ with lower mean/p50 latency, because a saturated index hit ratio eliminates metadata-induced disk I/O and outweighs the lower data-block hit rate.

**Trade-off at large cache sizes.** At $\geq 350 \times 10^3$, $r = 0.55$ is best: the cache now holds all index/filter blocks regardless of the cap, so a higher data-block quota additionally improves data hit rate. Thus $r$ should track the cache budget; the offline scheduler (Section 4.2.2) can update $r$ jointly with $q_w^L$.

# 8 RELATED WORKS

## 8.1 RDMA-Based Data Systems

A substantial body of work redesigns databases, key-value stores, and indexes around RDMA. FaRM [8], DrTM [45], and FORD [27]

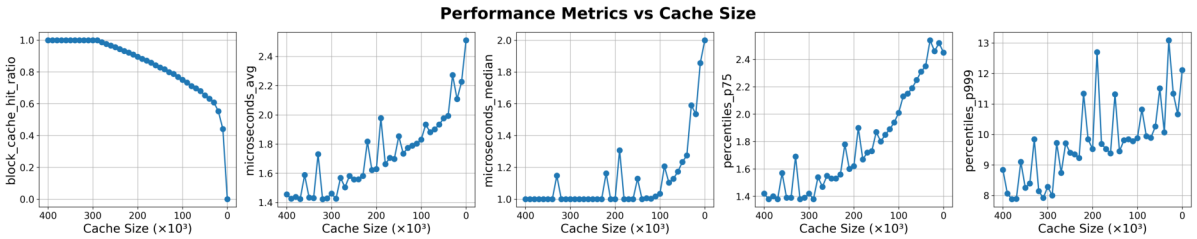



**Figure 8: Data access performance under different cache size.**

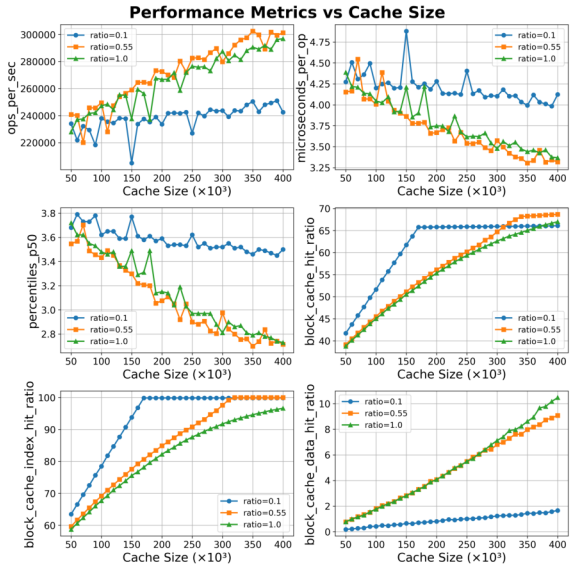



**Figure 9: Data access performance under different cache management policies.**

build distributed transactions over one-sided RDMA (FaRM/FORD) or combine RDMA with HTM (DrTM). Pilaf [30], FaSST [16], and HERD [14] span the design space of RDMA key-value stores, trading off one-sided reads, two-sided messaging, and hybrid server-side processing. GAM [5] and Clover [47] propose coherent global address spaces and metadata–data disaggregation, respectively. RDMA-optimized indexes include Sherman [44], Mira [13], and ROLEX [50].

All of these target single-tenant trusted clusters and redesign the full data path around RDMA. Narwhal instead operates in a multi-tenant cloud environment requiring hardware-enforced isolation (Section 5.2) and per-tenant QoS (Section 5.3), and integrates remote memory as an elastic cache tier within an existing OLAP engine to minimize adoption cost.

## 8.2 Remote Memory Management

Infiniswap [11], Fastswap [1], and Leap [28] expose remote DRAM as an OS-level swap device with increasingly sophisticated kernel integration and prefetching; Carbink [55] adds erasure-coded fault tolerance. At the pooling level, Spirit [23], Pond [12], Canvas [42], TPP [29], and DirectCXL [10] explore page-coherent, CXL-based, or hybrid RDMA/CXL abstractions.

These systems maximize compatibility by being transparent to applications but treat memory as semantically opaque. In contrast, Narwhal's Memory Allocator (Section 6) leverages the database engine's knowledge of block types and access patterns to keep latency-critical metadata local and sink tolerant data blocks to remote memory.

## 8.3 Multi-Tenant Isolation in Disaggregated Memory

Multi-tenant isolation in RDMA-based memory pools remains underexplored. 1RMA [37] re-envisions remote memory access for multi-tenant datacenters by introducing a new RDMA primitive that decouples naming from protection, enabling flexible sharing policies; however, it requires custom NIC hardware modifications not yet available on commodity RNICs. TDMem [46] provides trusted disaggregated memory with hardware-assisted encryption and access control trusted execution environments (TEEs), but targets security rather than performance isolation and QoS among database tenants. PolarDB Serverless [52] and PolarDB-MP [53] separate compute from shared memory or storage in a cloud database context; they provide instance-level isolation but are too coarse to protect workers within the same instance competing for shared remote memory.

Narwhal occupies a unique point in this design space: it provides hardware-enforced tenant isolation via standard RDMA Protection Domains on commodity NICs (Section 5.2), layered with per-tenant memory quotas, weighted fair queuing, and bandwidth throttling (Section 5.3)—all integrated with a database-aware memory allocator and a joint resource scheduler. To our knowledge, Narwhal is the first system to combine all three properties (hardware isolation, application-aware placement, and joint scheduling) for multi-tenant analytical databases over RDMA-disaggregated memory.

# 9 CONCLUSION

In this paper, we presented Narwhal, an RDMA-co-designed analytical database that overcomes the rigid per-machine memory boundary in multi-tenant OLAP environments. It combines a joint scheduler, hardware-enforced tenant isolation via RDMA Protection Domains, and a database-aware three-tier memory allocator. Deployed in production on Alibaba AnalyticDB, Narwhal eliminates OOM incidents, reduces TCO by 21.54%, and boosts memory overselling from 92.67% to 121.89% (+29.22)—without database migration or SLO degradation.

## REFERENCES


[1] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2020. Can far memory improve job throughput?. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 14, 16 pages. https://doi.org/10.1145/3342195.3387522

[2] Rahul Arora, Arnab Bhattacharya, and Pranab Sen. 2023. Adaptive Memory Management for Cloud Databases. *ACM Transactions on Database Systems* 48, 2 (2023), 1–35. https://doi.org/10.1145/3575678

[3] Bradley Barnhart, Marc Brooker, Daniil Chinenkov, Tony Hooper, Jihoun Im, Prakash Chandra Jha, Tim Kraska, Ashok Kurakula, Alexey Kuznetsov, Grant McAlister, Arjun Muthukrishnan, Aravinthan Narayanan, Douglas Terry, Bhuvan Urgaonkar, and Jiaming Yan. 2024. Resource management in Aurora Serverless. In *VLDB 2024*. https://www.amazon.science/publications/resource-management-in-aurora-serverless

[4] Noman Bashir, Nan Deng, Krzysztof Rzadca, David Irwin, Sree Kodak, and Rohit Jnagal. 2021. Take it to the limit: peak prediction-driven resource oversubscription in datacenters. In *Proceedings of the Sixteenth European Conference on Computer Systems* (Online Event, United Kingdom) *(EuroSys '21)*. Association for Computing Machinery, New York, NY, USA, 556–573. https://doi.org/10.1145/3447786.3456259

[5] Ayon Basumallik and Rudolf Eigenmann. 2006. Optimizing irregular shared-memory applications for distributed-memory systems. In *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York, New York, USA) *(PPoPP '06)*. Association for Computing Machinery, New York, NY, USA, 119–128. https://doi.org/10.1145/1122971.1122990

[6] David Breitgand and Amir Epstein. 2012. Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds. In *2012 Proceedings IEEE INFOCOM*. 2861–2865. https://doi.org/10.1109/INFCOM.2012.6195716

[7] Alibaba Cloud. [n.d.]. AnalyticDB for MySQL. https://www.alibabacloud.com/product/analyticdb-for-mysql

[8] Alejandro Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 401–414. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/dragojevic

[9] Peter X. Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2016. Network Requirements for Resource Disaggregation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 249–264. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gao

[10] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. 2022. Direct Access, High-Performance Memory Disaggregation with DirectCXL. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 287–294. https://www.usenix.org/conference/atc22/presentation/gouk

[11] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. 2017. Efficient Memory Disaggregation with Infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 649–667. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu

[12] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. 2021. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. ACM, Virtual Event, 388–403. https://doi.org/10.1145/3477132.3483570

[13] Zhipeng Jia, Anna Eusebi, Tyler Harter, Anurag Khandelwal, Jiacheng Wu, Lilia Budke, Venkatraman Govindaraju, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2015. Mira: A Key-Value Cache for Fast RDMA-based Ordered Key-Value Lookups. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Santa Clara, CA, 159–173. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/jia

[14] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2014. Using RDMA efficiently for key-value services. *SIGCOMM Comput. Commun. Rev.* 44, 4 (Aug. 2014), 295–306. https://doi.org/10.1145/2740070.2626299

[15] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. Design Guidelines for High Performance RDMA Systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, Denver, CO, 437–450. https://www.usenix.org/conference/atc16/technical-sessions/presentation/kalia

[16] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. FaSST: fast, scalable and simple distributed transactions with two-sided (RDMA) datagram RPCs. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 185–201.

[17] Mingyang Kong, Hehe Jiang, Yi Li, Fengyuan Xu, Idan Yaniv, and Cheng Li. 2023. Understanding RDMA Microarchitecture Resources for Performance Isolation. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 75–93. https://www.usenix.org/conference/nsdi23/presentation/kong

[18] Arnd Christian König, Yi Shan, Tobias Ziegler, Aarati Kakaraparthy, Willis Lang, Justin Moeller, Ajay Kalhan, and Vivek Narasayya. 2022. Tenant placement in over-subscribed database-as-a-service clusters. *Proc. VLDB Endow.* 15, 11 (jul 2022), 2559–2571. https://doi.org/10.14778/3551793.3551814

[19] Kubernetes. [n.d.]. Kubernetes. https://kubernetes.io/

[20] Alok Gautam Kumbhare, Reza Azimi, Ioannis Manousakis, Anand Bonde, Felipe Frujeri, Nithish Mahalingam, Pulkit A. Misra, Seyyed Ahmad Javadi, Bianca Schroeder, Marcus Fontoura, and Ricardo Bianchini. 2021. Prediction-Based Power Oversubscription in Cloud Platforms. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 473–487. https://www.usenix.org/conference/atc21/presentation/kumbhare

[21] Arnd Christian König, Yi Shan, Karan Newatia, Luke Marshall, and Vivek Narasayya. 2023. Solver-In-The-Loop Cluster Resource Management for Database-as-a-Service. In *50th International Conference on Very Large Databases*. VLDB Endowment, Inc. https://www.microsoft.com/en-us/research/publication/solver-in-the-loop-cluster-resource-management-for-database-as-a-service/

[22] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, Greg Thelen, Kamil Adam Yurtsever, Yu Zhao, and Parthasarathy Ranganathan. 2019. Software-Defined Far Memory in Warehouse-Scale Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 317–330. https://doi.org/10.1145/3297858.3304053

[23] Sangmin Lee, Juncheng Gu, Youngmoon Lee, Mosharaf Chowdhury, and Kang G. Shin. 2021. SPIRIT: Improving the Scalability of Distributed Shared Memory by Reducing Working Set Size. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. ACM, Virtual Event, 353–368. https://doi.org/10.1145/3477132.3483568

[24] Youngmoon Lee, Hasan Al Maruf, Mosharaf Chowdhury, Asaf Cidon, and Kang G. Shin. 2022. Hydra : Resilient and Highly Available Remote Memory. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*. USENIX Association, Santa Clara, CA, 181–198. https://www.usenix.org/conference/fast22/presentation/lee

[25] Ji You Li, Jiachi Zhang, Yuhang Liu, Wenchao Zhou, Xin Zhou, Fangyuan Zhou, and Feifei Li. 2025. Eigen+: Memory Over-Subscription for Alibaba Cloud Databases. In *Companion of the 2025 International Conference on Management of Data* (Berlin, Germany) *(SIGMOD/PODS '25)*. Association for Computing Machinery, New York, NY, USA, 525–538. https://doi.org/10.1145/3722212.3724435

[26] Ji You Li, Jiachi Zhang, Wenchao Zhou, Yuhang Liu, Shuai Zhang, Zhuoming Xue, Ding Xu, Hua Fan, Fangyuan Zhou, and Feifei Li. 2023. Eigen: End-to-End Resource Optimization for Large-Scale Databases on the Cloud. *Proc. VLDB Endow.* 16, 12 (aug 2023), 3795–3807. https://doi.org/10.14778/3611540.3611565

[27] Haodi Lu, Haikun Liu, Yujian Zhang, Zhuohui Duan, Xiaofei Liao, Hai Jin, and Yu Zhang. 2025. Fast distributed transactions for RDMA-based disaggregated memory. In *Proceedings of the 2025 USENIX Conference on Usenix Annual Technical Conference* (Boston, MA, USA) *(USENIX ATC '25)*. USENIX Association, USA, Article 55, 16 pages.

[28] Hasan Al Maruf and Mosharaf Chowdhury. 2020. Effectively Prefetching Remote Memory with Leap. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 843–857. https://www.usenix.org/conference/atc20/presentation/al-maruf

[29] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2023. TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 742–755. https://doi.org/10.1145/3582016.3582063

[30] Christopher Mitchell, Yifeng Geng, and Jinyang Li. 2013. Using one-sided RDMA reads to build a fast, CPU-efficient key-value store. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference* (San Jose, CA) *(USENIX ATC'13)*. USENIX Association, USA, 103–114.

[31] Vikram Nathan, Vikramank Singh, Zhengchun Liu, Mohammad Rahman, Andreas Kipf, Dominik Horn, Davide Pagano, Gaurav Saxena, Balakrishnan Narayanaswamy, and Tim Kraska. 2024. Intelligent Scaling in Amazon Redshift. In *Companion of the 2024 International Conference on Management of Data* (Santiago AA, Chile) *(SIGMOD/PODS '24)*. Association for Computing Machinery, New York, NY, USA, 269–279. https://doi.org/10.1145/3626246.3653394

[32] James Pinkerton and Ellen Deleganes. 2007. Direct Data Placement Protocol (DDP) / Remote Direct Memory Access Protocol (RDMAP) Security. RFC 5042. https://doi.org/10.17487/RFC5042

[33] Oriana Riva, Sathiya Keerthi Murugan, Davide Frey, Alexandre Maurer, Antonio Paolillo, Anne-Marie Kermarrec, Eyal de Lara, and Willy Zwaenepoel. 2021.

CherryPick: Adapting Datacenter Operations to Peak Efficiency. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. ACM, Virtual Event, 404–420. https://doi.org/10.1145/3477132.3483571

[34] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, and John Wilkes. 2020. Autopilot: Workload Autoscaling at Google. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 16, 16 pages. https://doi.org/10.1145/3342195.3387524

[35] Gaurav Saxena, Mohammad Rahman, Naresh Chainani, Chunbin Lin, George Caragea, Fahim Chowdhury, Ryan Marcus, Tim Kraska, Ippokratis Pandis, and Balakrishnan (Murali) Narayanaswamy. 2023. Auto-WLM: Machine Learning Enhanced Workload Management in Amazon Redshift. In *Companion of the 2023 International Conference on Management of Data* (Seattle, WA, USA) *(SIGMOD '23)*. Association for Computing Machinery, New York, NY, USA, 225–237. https://doi.org/10.1145/3555041.3589677

[36] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 69–87. https://www.usenix.org/conference/osdi18/presentation/shan

[37] Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F. Wenisch, Monica Wong-Chan, Sean Clark, Milo M. K. Martin, Moray McLaren, Prashant Chandra, Rob Cauble, Hassan M. G. Wassel, Behnam Montazeri, Simon L. Sabato, Joel Scherpelz, and Amin Vahdat. 2020. 1RMA: Re-envisioning Remote Memory Access for Multi-tenant Datacenters. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 708–721. https://doi.org/10.1145/3387514.3405897

[38] Rebecca Taft, Nosayba El-Sayed, Marco Serafini, Yu Lu, Ashraf Aboulnaga, Michael Stonebraker, Ricardo Mayerhofer, and Francisco Andrade. 2018. P-Store: An Elastic Database System with Predictive Provisioning. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 205–219. https://doi.org/10.1145/3183713.3190650

[39] Jian Tan, Tieying Zhang, Feifei Li, Jie Chen, Qixing Zheng, Ping Zhang, Honglin Qiao, Yue Shi, Wei Cao, and Rui Zhang. 2019. iBTune: individualized buffer tuning for large-scale cloud databases. *Proc. VLDB Endow.* 12, 10 (jun 2019), 1221–1234. https://doi.org/10.14778/3339490.3339503

[40] Sean J. Taylor and Benjamin Letham. 2018. Forecasting at Scale. *The American Statistician* 72, 1 (2018), 37–45. https://doi.org/10.1080/00031305.2017.1380080

[41] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-Scale Cluster Management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems* (Bordeaux, France) *(EuroSys '15)*. Association for Computing Machinery, New York, NY, USA, Article 18, 17 pages. https://doi.org/10.1145/2741948.2741964

[42] Chenxi Wang, Yifan Qiao, Haoran Ma, Shi Liu, Wenguang Chen, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2023. Canvas: Isolated and Adaptive Swapping for Multi-Applications on Remote Memory. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 161–179. https://www.usenix.org/conference/nsdi23/presentation/wang-chenxi

[43] Lu Wang, Cheng-Zhong Xu, Zonghua Zhang, Jie Yang, Ming Zhao, and Yunhua Li. 2012. Stochastic Bin Packing for Resource Allocation in Cloud Computing. In *2012 IEEE 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2861–2865. https://doi.org/10.1109/INFCOM.2012.6195716

[44] Qing Wang, Youyou Lu, and Jiwu Shu. 2022. Sherman: A Write-Optimized Distributed B+Tree Index on Disaggregated Memory. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1033–1048. https://doi.org/10.1145/3514221.3517824

[45] Xingda Wei, Jiaxin Shi, Yanzhe Chen, Rong Chen, and Haibo Chen. 2015. Fast in-memory transaction processing using RDMA and HTM. In *Proceedings of the 25th Symposium on Operating Systems Principles* (Monterey, California) *(SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 87–104. https://doi.org/10.1145/2815400.2815419

[46] Ke Xia and Sheng Wei. 2024. DM-TEE: Trusted Execution Environment for Disaggregated Memory. In *Proceedings of the Great Lakes Symposium on VLSI 2024* (Clearwater, FL, USA) *(GLSVLSI '24)*. Association for Computing Machinery, New York, NY, USA, 204–209. https://doi.org/10.1145/3649476.3658702

[47] Ziwei Xiong, Dejun Jiang, and Jin Xiong. 2024. DiStore: A Fully Memory Disaggregation Friendly Key-Value Store with Improved Tail Latency and Space Efficiency. In *Proceedings of the 53rd International Conference on Parallel Processing* (Gotland, Sweden) *(ICPP '24)*. Association for Computing Machinery, New York, NY, USA, 607–617. https://doi.org/10.1145/3673038.3673088

[48] Jie Yan, Yunlei Lu, Liting Chen, Si Qin, Yixin Fang, Qingwei Lin, Thomas Moscibroda, Saravan Rajmohan, and Dongmei Zhang. 2022. Solving the Batch Stochastic Bin Packing Problem in Cloud: A Chance-constrained Optimization Approach. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) *(KDD '22)*. Association for Computing Machinery, New York, NY, USA, 2169–2179. https://doi.org/10.1145/3534678.3539334

[49] Renyu Yang, Chunming Hu, Xiaoyang Sun, Peter Garraghan, Tianyu Wo, Zhenyu Wen, Hao Peng, Jie Xu, and Chao Li. 2020. Performance-Aware Speculative Resource Oversubscription for Large-Scale Clusters. *IEEE Transactions on Parallel and Distributed Systems* 31, 7 (2020), 1499–1517. https://doi.org/10.1109/TPDS.2020.2970013

[50] Xiangyao Yu, George Bezerra, Andrew Pavlo, Srinivas Devadas, and Michael Stonebraker. 2015. Staring into the Abyss: What Happens When a New Concurrency Control Algorithm Meets a Diverse Set of Real-world Workloads?. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, Melbourne, Victoria, Australia, 453–467. https://doi.org/10.1145/2735471.2735478

[51] Chaoqun Zhan, Maomeng Su, Chuangxian Wei, Xiaoqiang Peng, Liang Lin, Sheng Wang, Zhe Chen, Feifei Li, Yue Pan, Fang Zheng, and Chengliang Chai. 2019. AnalyticDB: Real-Time OLAP Database System at Alibaba Cloud. *Proc. VLDB Endow.* 12, 12 (aug 2019), 2059–2070. https://doi.org/10.14778/3352063.3352124

[52] Ce Zhang, Feifei Li, Junjie Zhou, Weiwei Wang, Zhongle Xie, Yijian Wang, Kun Li, Qiong Luo, Wei Zhao, Nengwei Zhang, Lin Zong, Shuncheng Liu, Junchen Zhang, Qingqing Chen, Jiaqi Liu, Jiayi Wang, Haowen Lin, Zhihui Wei, Fenfang Liang, Ying Li, Yifan Wang, Yiming Zhang, Tiancheng Wu, Lingzhi Geng, Xiaomeng Zhang, Jiguang Han, Lu Chen, Pengfei Zhang, and Wei Li. 2024. PolarDB Serverless: A Cloud Native Database for Massively Scaling Out. In *Proceedings of the 2024 ACM SIGMOD International Conference on Management of Data*. ACM. https://doi.org/10.1145/3639266

[53] Junchen Zhang, Ce Zhang, Feifei Li, Yijian Wang, Junjie Zhou, Weiwei Wang, Zhongle Xie, Qiong Luo, Wei Zhao, Nengwei Zhang, Lin Zong, Shuncheng Liu, Qingqing Chen, Jiaqi Liu, Jiayi Wang, Haowen Lin, Zhihui Wei, Fenfang Liang, and Ying Li. 2023. PolarDB-MP: A Multi-Proxy Architecture for Cloud-Native Databases. In *Proceedings of the 2023 ACM SIGMOD International Conference on Management of Data*. ACM. https://doi.org/10.1145/3639266

[54] Cheng Zhou, Jiacheng Zhang, Jian Zhou, Yang Zhang, and Yongbo Wang. 2022. Flux: A Demand-Aware Scheduler for Cloud Data Warehouses. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM. https://doi.org/10.1145/3514221.3520123

[55] Yang Zhou, Hassan M. G. Wassel, Sihang Liu, Jiaqi Gao, James Mickens, Minlan Yu, Chris Kennelly, Paul Turner, David E. Culler, Henry M. Levy, and Amin Vahdat. 2022. Carbink: Fault-Tolerant Far Memory. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 55–71. https://www.usenix.org/conference/osdi22/presentation/zhou-yang