# 基于UMX统一内存/存储扩展的AI Scale Up架构

# AI Scale-Up Architecture Based on UMX Unified Memory/Storage Extension

阿里云 超高速互连负责人 孔阳

# Agent推理时代的挑战

## Agent 任务效果

- 需要支持多种工具调用（API、代码执行、数据库、搜索等）
- 单个任务可能需要多步骤串行完成
- 复杂场景需多个子 Agent 协同分工，对调度与编排能力要求高

## 用户体验

- 需要对用户的长期习惯和过往历史有良好的理解力
- 执行复杂任务、多任务并发时的响应速度与输出速度
- 典型案例：AI Coding — 长上下文 + 实时代码补全，对延迟极度敏感

## Token 成本

- 单个 Credit 价格由多个因素组成：模型推理算力、上下文长度、工具调用链路、内存/存储访问等
- Agent 每轮自主决策可能涉及多次工具调用和长上下文推理，单次任务的 Credit 消耗远超普通对话
- 需要在任务效果与成本效率之间找到最优平衡

Agent 挑战不仅仅是 GPU 算力问题，更是多种算力、内存/存储功能协同的架构设计挑战

# AI 工程演进路径与记忆能力

## 01 对话

形态
单轮问答，无上下文积累

记忆能力
仅当前会话的短期上下文窗口

典型场景
简单 QA、闲聊

## 02 提示词工程

形态
System Prompt、Few-shot、CoT 引导模型

记忆能力
经验与规则固化在提示词模板中（静态记忆）

典型场景
角色扮演、结构化输出

## 03 Agent

形态
自主规划、工具调用、多步骤完成任务

记忆能力
短期工作记忆 + 工具调用历史；跨会话仍依赖外部

典型场景
AI Coding、自动化工作流

## 04 Harness

形态
为 Agent 提供结构化运行环境与状态管理

记忆能力
持久化任务状态、工具链路追踪、多 Agent 上下文共享

典型场景
LangGraph、CrewAI、AutoGen

## 05 Others

形态
RAG / Fine-tuning / Memory 系统等

记忆能力
外部知识库长期记忆 + 参数内化 + 专用记忆层（向量库+摘要+时间衰减）

典型场景
企业知识库、个性化助手

记忆能力不断增强：短期窗口 → 静态固化 → 工作记忆 → 持久化状态 → 长期记忆系统

核心洞察：AI 工程的演进本质是不断扩展"记忆"的边界——从窗口内短期记忆，到提示词固化记忆，到 Agent 工作记忆，再到外部持久化长期记忆，最终实现类人的记忆-推理协同

# 多种内存-计算优化方向

## KV Cache 内存管理

GPU 侧效率优化

- PagedAttention (SOSP 2023)：KV Cache 分页管理，类似 OS 虚拟内存，消除碎片和冗余拷贝，near-zero 浪费
- KIVI (ICML 2024)：2-bit 非对称量化（Key 按 channel / Value 按 token），显存占用压缩至 1/4，无需调参
- RadixAttention / SGLang (2024)：Radix 树自动缓存复用相同前缀的 KV Cache，避免多请求重复计算，吞吐 up to 5x

## 跨存储层级卸载

GPU → CPU → SSD

- FlexGen (ICML 2023)：GPU → CPU DRAM → SSD 三层卸载 + 4-bit 量化 + zig-zag 调度，175B 模型单卡吞吐提升 100x
- InfiniGen (OSDI 2024)：KV Cache 池置于 CPU 端，投机预取按注意力分数动态换入 GPU，长文本加速 up to 3x
- Mooncake (FAST 2025)：KV Cache 为中心的分布式架构，复用 GPU 节点闲置 CPU/DRAM/SSD 构建分布式缓存，吞吐 +525%
- HCache (EuroSys 2025)：从中间层激活恢复 LLM 状态，避免完整重算，TTFT 降低 up to 5.73x，存储开销 -1.92~2.40x

## 计算-存储解耦

分离计算与记忆

- Engram (DeepSeek, 2026)：20-25% 参数为静态记忆模块，外置到 host DRAM，O(1) 查找，验证 100B 参数 embedding 表卸载可行
- Splitwise (ISCA 2024)：Prefill (计算密集) 和 Decode (带宽/容量密集) 分到不同硬件池，KV Cache 异步传输，同预算吞吐 +2.35x

# KV Cache容量需求增加迅猛

### 1. 长上下文推理

- LLaMA 7B 64K → ~64 GB
- 10M context → TB 级
- 线性增长，远超模型权重

### 2. AI Agent 多轮对话

- Prefix Cache 命中率 95%+
- 工具调用放大效应 3x
- 上下文膨胀，NIC 带宽饱和

### 3. 多模态大模型

- 视觉 token 是文本的 10-100x
- MEDA KV 缩减 72%，速度提升 2.82x
- LMCache TTFT 50s → 3s

### 4. 代码生成与编程助手

- >10K tokens 即可压垮 GPU HBM
- 代码库上下文可达百万 tokens
- 结构化 KV 压缩需求迫切

### 5. RAG 检索增强

- ~1 MB / token KV Cache 消耗
- CXL 内存扩展 Batch +30%
- 多级存储（HBM→DRAM→SSD）刚需

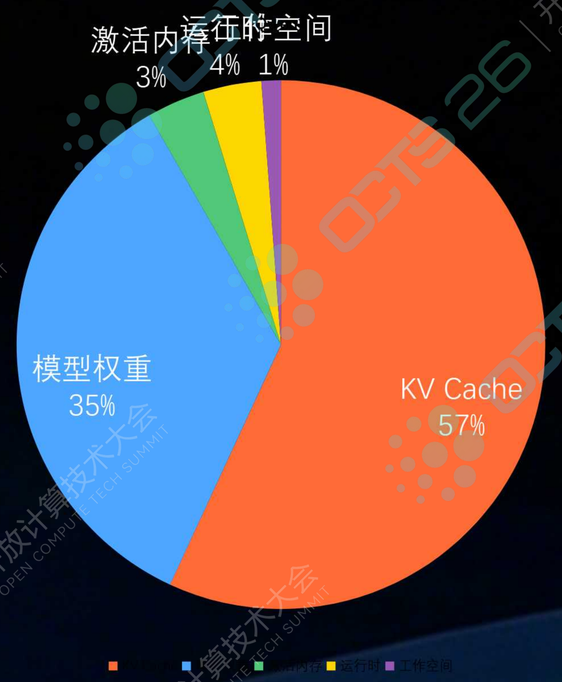
**不同场景总 KV Cache 需求对比**

| 场景 | 配置 | 总 KV Cache 需求 |
|---|---|---|
| 单请求短文本 | 7B 模型, 512 tokens | ~0.5 GB |
| 单请求长文本 | 7B 模型, 4K tokens | ~2 GB |
| 超长上下文 | 7B 模型, 64K tokens | ~64 GB |
| 批量推理 | 70B 模型, 8K, batch=32 | ~640 GB |
| Agent 多轮 | 含 5 个工具调用 | 有效需求增加 3 倍 |
| 多模态视频 | 高分辨率多帧 | 文本场景的 10-100 倍 |

# HBM内存占用-KV Cache需要外置池化扩展

| 内存类别 | 说明 | Qwen3-235B | DeepSeek-V3 | 带宽需求 |
|---|---|---|---|---|
| **模型权重** | 全量参数FP8<br>(TP8每卡) | 235GB<br>(29GB/卡) | 671GB<br>(84GB/卡) | **极高** |
| **KV Cache**<br>(1req×32K) | GQA/MLA压缩<br>K/V状态缓存 | 单任务百GB+，<br>需要极大的HBM外池化内存<br>进行KV Cache命中提升 | | **高** |
| **KV Cache**<br>(1req×128K) | 长上下文场景 | | | |
| **KV Cache**<br>(64并发×32K) | 高吞吐服务<br>总池容量 | | | |
| **激活内存** | 前向中间张量<br>(per-req, TP分摊) | 1~4 GB/卡 | 1~4 GB/卡 | **极高** |
| **运行时开销** | CUDA Ctx+NCCL<br>+内存碎片 | 2~4 GB/卡 | 2~4 GB/卡 | **低** |
| **工作空间** | GEMM+Sampling<br>+页表 | 0.5~2 GB/卡 | 0.5~2 GB/卡 | **高** |

**BHM上内存占比分布 (64并发×32K)**



# 性能需求与超节点优势

| 应用场景 | TTFT 典型值 | TPOT 典型值 | 特征 |
| --- | --- | --- | --- |
| 人对话 (Chat Q&A) | 0.05 ~ 2 s | 20 ~ 30 ms | 输入短、容忍首 Token 延迟 |
| 代码生成 (Coding) | **2 ~ 19 s** | 5 ~ 30 ms | 长上下文 + 思维链推理 |
| 多 Agent 协同 | **×N 累积** | ×N 累积 | N 轮串行调用延迟叠加 |

## 超节点多 GPU 深度协同对 Agent TTFT / TPOT 的优势

### Tensor Parallel
### Prefill 加速

N 卡并行计算 Attention
→ TTFT 近线性缩短
8 卡 → TTFT 降至 1/6~1/8

### Expert Parallel
### 免跨节点通信

超节点内 All-to-All
带宽 TB/s 级
→ MoE 路由零等待

### 多轮累积
### 延迟收敛

TTFT 1/N → Agent N 轮
总延迟 ≈ 单卡 1 轮
→ Agent 体验逼近 Chat

# 超节点多部署大模型的优势

## 01 模型全量驻留 · 零 Offload

- Tensor Parallel 将权重均匀切分到 N 张卡的 HBM
- 消除 CPU/SSD offload，推理全程 GPU 计算
- 2T MoE 模型 INT4 量化 ≈ 500GB → 8×80G 卡恰好承载

## 02 TTFT 降低 · Prefill 线性加速

- 多卡并行执行 Attention + FFN 前向计算
- Prefill 算力按 GPU 数线性叠加

## 03 TPOT 降低 · Decode 轻量高速

- Decode 每步每卡仅计算 1/N 参数量的矩阵乘法
- 超节点内 AllReduce 延迟 < 10μs（NVLink 900GB/s）
- 单 Token 生成时间压缩至 5-15 ms

## 04 超节点 vs 跨节点：延迟差距

- 超节点内：NVLink / UALink 延迟 1-5 μs，带宽 900 GB/s+
- 跨节点：RDMA 延迟 5-20 μs，带宽 50-100 GB/s
- 超节点 TP 效率 > 95%；跨节点 TP 效率骤降至 60-70%

**超节点内多个GPU的高算力、深度协同、大容量内存成为基础竞争力**

# 阿里云磐久UMX：AI 总线级存算互连架构

Unified Memory Extension

## 三大驱动：互连芯片+介质创新+软件系统


**Scale-Up 思路**
基于计算总线构建超节点
内存/存储池

**原生 AI 架构**
题覆性方案，GPU/CPU超节点
解决容量+成本+性能

**协议基础**
UALink + CXL 计算总线
内存语义 Load/Store 透明

**极致性能**
高带宽、低延迟
消除「存储墙」

**目标场景**
极速AI推理 · Agent记忆
超长KV Cache

极致性能与存算深度耦合 — 通过超节点 Scale Up 高带宽互连，将内存/存储池以纳秒级延迟统一接入算力节点，消除 AI Agent 时代的「存储墙」

# 阿里云磐久服务器发展历程

## PCIe 为中心的服务器

HPC/渲染等

### 直连 or 基于 PCIe Switch 的单机 up to 8卡

- 单卡各自独立作为协处理器：HPC/渲染
- 基于PCIe Switch扩展：PCIe Switch的P2P；带宽低，跨CPU延迟；Balanced Topology；虚拟化；DP并行

## 单机 Scale Up

AlexNet

ResNet

Perception AI

大 dataset，数据并行

### 单机 Scale Up 8卡 -> 16卡

- 多卡低延迟，高带宽互连，突破单芯片限制聚合成大算力，大显存“大卡”
- 芯片直出连接，基于典型Scale Up协议如NVLINK，UALINK
- Scale Up Domain较小，24年推出16卡才刚满足单机部署满血Deepseek 671B模型

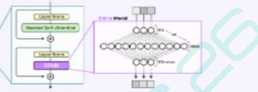
## 单机扩大 Scale Up 域

ChatGPT

Generative AI

大 model size，模型并行

## Scale Up 超节点



Agentic AI

MoE 混合专家模型，reasoning: 100X算力需求

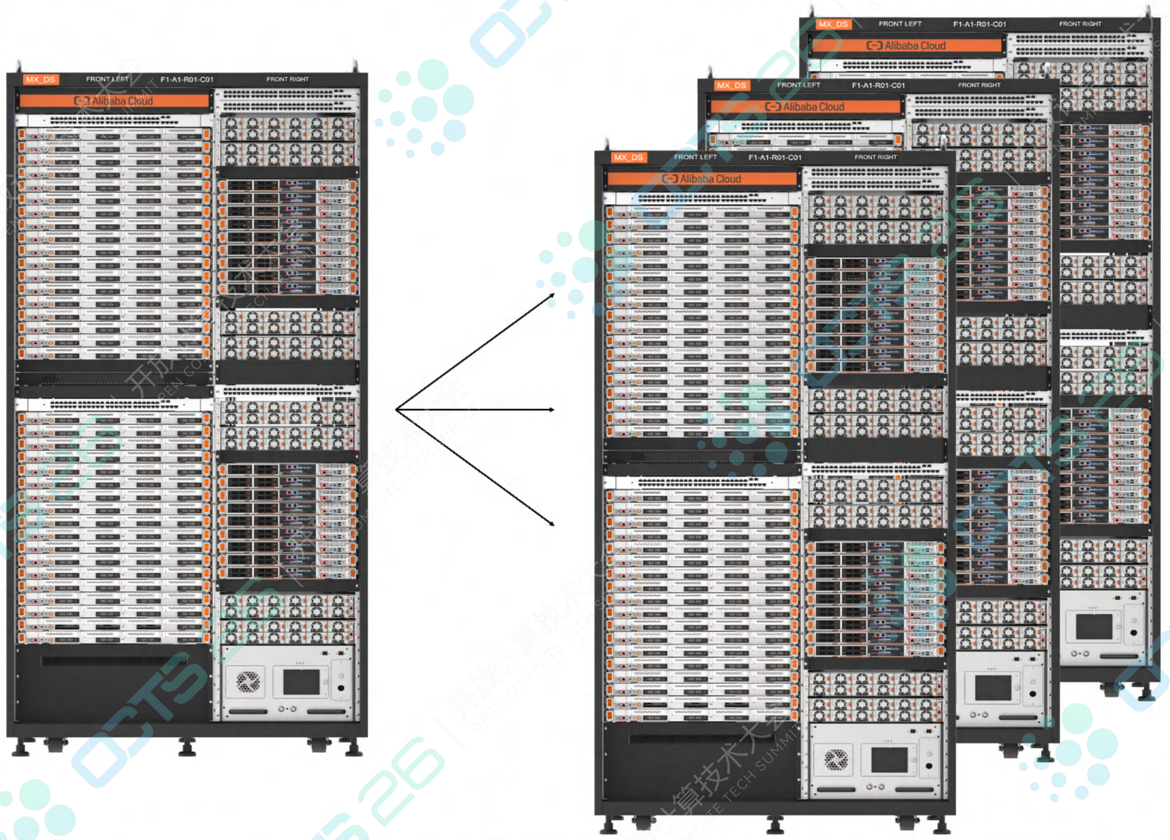
### 超大 Scale Up 域

- 柜级，超高密度
- 超高功率，液冷
- 高可靠设计

# 阿里云磐久超节点AL128

Prefill阶段常采用首Token延迟
(Time to first token，TTFT)作为SLO

Decode阶段采用输出Token延迟
（Time per output token, TPOT）作为SLO

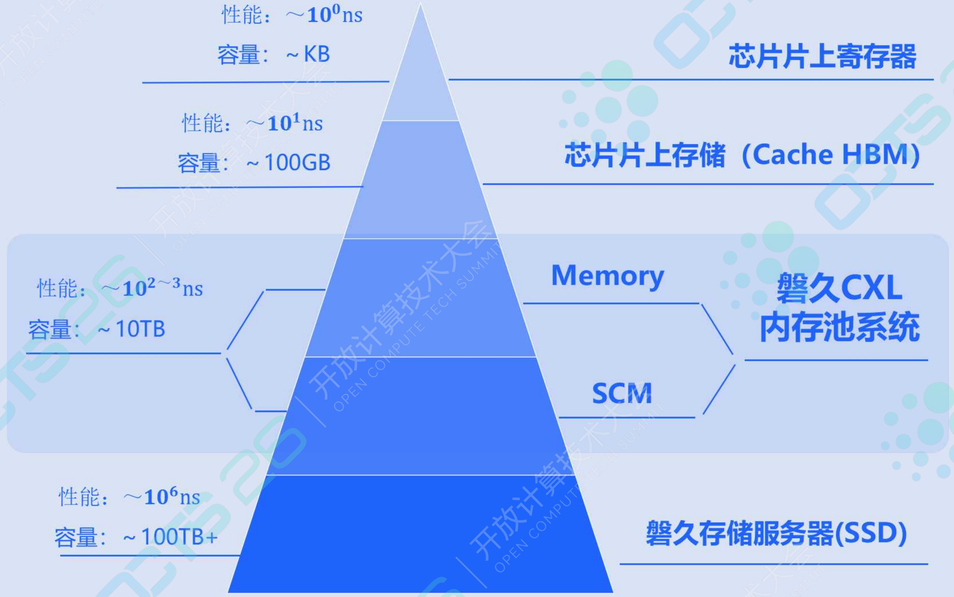



超节点架构：优秀的SLO和TCO

| 维度 | 多台8卡服务器 | 磐久超节点 |
| --- | --- | --- |
| 有效显存容量 | 碎片化限制。长文本需跨机切分，受限于外部网络带宽和同步开销。单次最大处理长度受限。 | 巨型统一池。可视为一张超大显卡，支持 单次完整加载 百万级 Token 上下文。 |
| MoE 专家路由效率 | 跨网延迟高<br>若专家分散在不同 8 卡节点，Token 路由需走外部网络 (IB/RoCE)，All-to-All 延迟高，严重拖慢生成速度。 | 片内极速交换<br>所有专家均在超节点域内。Token 分发如同内存拷贝，All-to-All 延迟降低大幅降低。 |
| 张量并行 (TP) 扩展性 | 跨机效率低<br>TP>8 时，每生成一个 Token 都要进行跨机 All-Reduce，通信延迟占主导，TPOT 随 TP 增大而恶化。 | 线性扩展至 72<br>TP=72 时，通信几乎无感。大模型可切分得更细，单卡显存压力更小，从而容纳更大 Batch。 |
| 首字延迟 (TTFT) | 随着 Prompt 变长，跨机同步开销占比增加。100k tokens 可能需要数秒甚至十数秒才能出首字。 | 亚秒级响应<br>利用聚合带宽和算力，FlashAttention 在卡间无缝执行。100k tokens 可实现 <1 秒 首字响应。 |
| 动态 Batch Size | 受限于单节点容量<br>单个 8 卡节点能容纳的并发 Prompt 数量有限。大 Batch 需跨节点调度，管理复杂且易负载不均。 | 超大 Batch 吞吐<br>可在一个逻辑单元内同时处理 数百个 长 Prompt 请求。<br>负载均衡由硬件拓扑自动优化。 |

# UMX：磐久CXL内存池化服务器

从芯片到系统，内存级全链路的创新



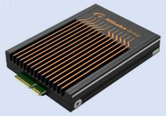

## CXL Switch节点，高性能，高灵活

- 高带宽(+10Tb/s)、低时延(<100ns)
- 端口灵活，上下行带宽收敛比可配
- 可适配磐久各类计算机型

## JBOM内存池节点，可搭载多个CXL模组

- 搭载E3.S形态CXL部件模组
- 超大容量、支持内存/非易失存储扩展
- 高运维便捷性

## AliMemory，CXL接口内存模组



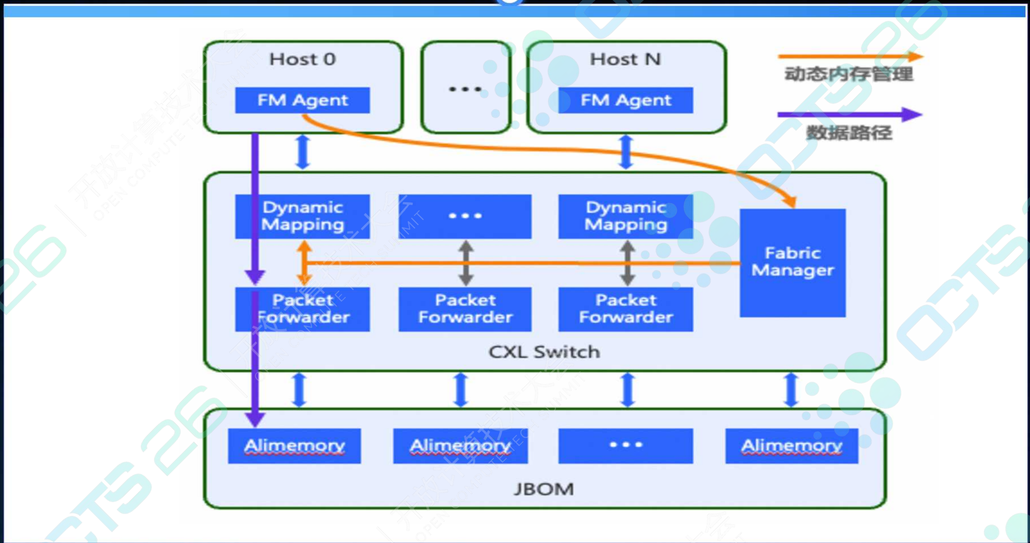
- 高带宽32GT/S，超低延迟
- 高RAS特性，稳定可靠
- 智能运维系统支持故障诊断、预测

## AliSCM CXL接口非易失介质


- 内存语义、持久化特性
- 自研核心IP、自研介质管理算法，
- 超高带宽，超低延迟，超大容量

# UMX：CXL数据面+管控面全栈方案

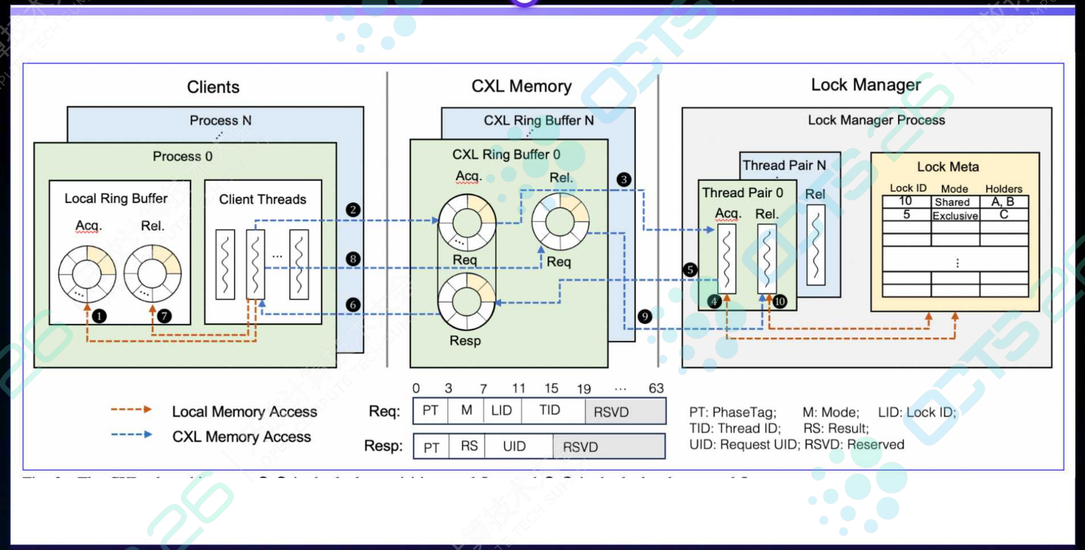
## 弹性




自研管控系统，细粒度资源管理，实时弹性伸缩

- ✓ 秒级弹性内存伸缩
- ✓ 计算和内存解耦，灵活扩容
- ✓ 池化共享，最大化内存利用率
- ✓ 高可用，内存故障独立运维

## 性能



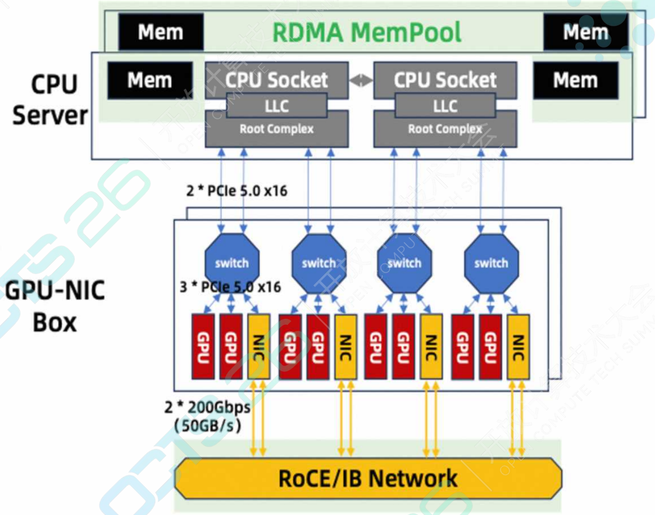

全套数据搬移与跨节点锁方案

- ✓ 套数据搬移在[64B, 16KB]区间有显著延迟优势
- ✓ 跨节点锁方案: CXLock: Efficient and Scalable Lock Management for CXL-enabled Distributed Systems @TC 2026

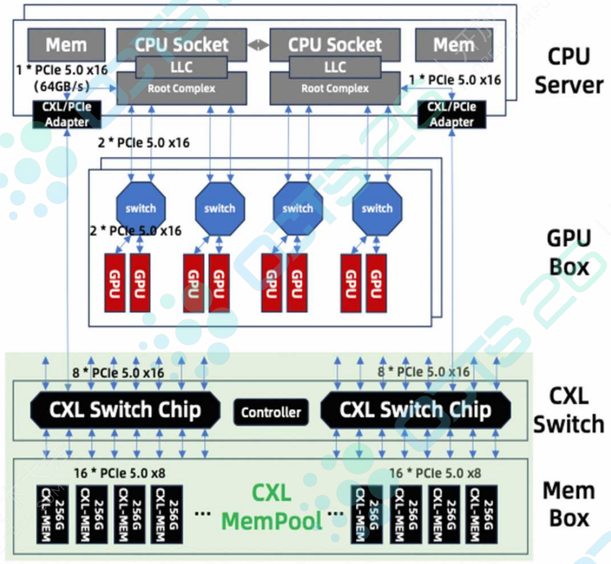
# UMX CXL内存池AI KV Cache应用：首Token延迟直降89.6%

## 阿里云提出基于CXL的KV缓存管理内存架构Beluga

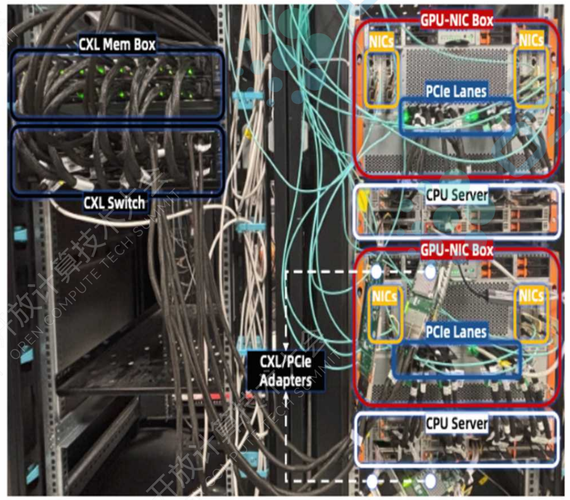
- Beluga: A CXL-Based Memory Architecture for Scalable and Efficient LLM KVCache Management
- 使GPU和CPU能够通过CXL交换机访问共享的大规模内存池，构建Beluga-KVCache系统
- 与基于RDMA的方案相比，Beluga-KVCache的首Token延迟（TTFT）降低了89.6%，vLLM吞吐量提升了7.35倍




(a) GPU clusters with an RDMA-based memory pool



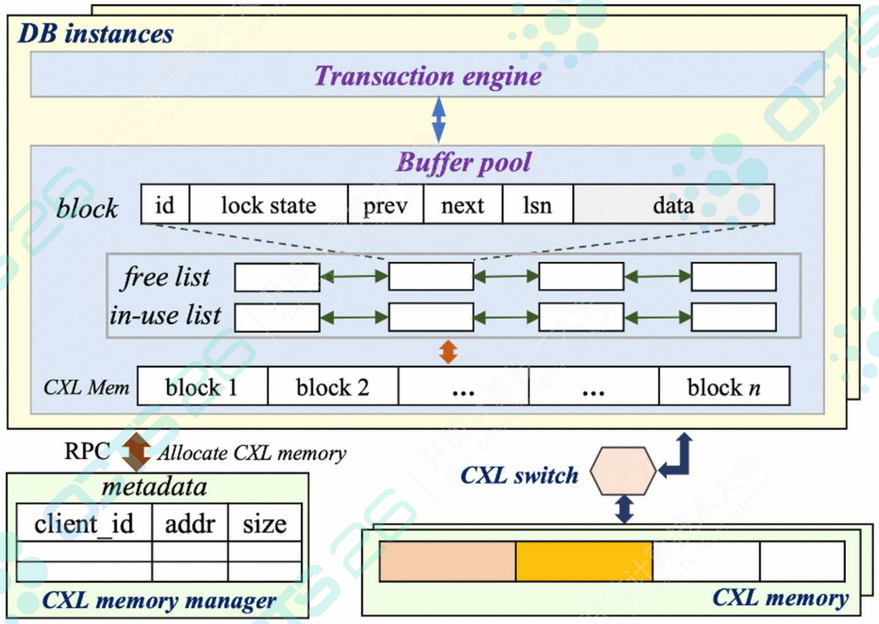

(b) GPU clusters with a CXL-based memory pool (*Beluga*)



# UMX：CXL内存池PolarDB应用：内存三层解耦

## CXL赋能PolarDB，打造新一代分布式内存池

- 业界首个基于CXL Switch的云数据库应用落地案例，基于内存池重构业务引擎，打破了传统RDMA在带宽、延迟、恢复效率等方面的技术瓶颈
- SIGMOD'25 工业赛道最佳论文的《Unlocking the Potential of CXL for Disaggregated Memory in Cloud-Native Databases》




- CXL内存池内构建Buffer Pool
- 本地内存只用于事务引擎
- 主机缓存一致性软件方案
- 基于MemPool跨主机通信

OPEN Compute Project®

开放计算标准工作委员会 Open Compute Technology Committee

# THANKS