arXiv:2611.32083v1 [cs.NI] 24 Nov 2026

面向分布式 MoE 推理的推理感知拥塞控制

Weiye Chen*†, Qi Xin*, Ahmed M. Abdelmoniem†
*北京邮电大学，北京，中国
†伦敦玛丽女王大学，伦敦，英国

摘要—拥塞控制对于实现低延迟数据中心通信至关重要，但链路利用率和队列状态无法表明已交付的字节是否推动了应用程序进展。这一区别在分布式混合专家（MoE）推理中尤为重要：集合传输共同满足 GPU 依赖关系，而计算可能与通信重叠。因此，限制消费者进度的传输可能会随时间变化，并且在该消费者仍处于计算阶段时，额外带宽可能几乎不起作用。我们研究了这种交互，并提出 IACC：在分配队列对（QP）速率时，将 ACK/INT 反馈与集合通信进度和消费者进度相结合。固定快照分析刻画了预算守恒和加权最大—最小分配。在基线到达密度下的五种跟踪引导工作负载中，与 HPCC 相比，IACC 将平均首令牌时间和每输出令牌时间分别降低了 6.32%–19.41% 和 7.02%–27.33%，在 1.25×涵盖的测试工作负载—密度设置中，平均首令牌时间和第 95 百分位首令牌时间也更低。

索引词—**拥塞控制、分布式推理、集体通信、混合专家。**

I. 引言

端到端请求延迟是分布式推理中服务质量（QoS）的核心问题。在混合专家（MoE）服务中，请求会反复在 GPU 之间交换数据，以分发令牌并组合专家输出。这些传输承载着后续计算所需的输入，因此通信延迟可能延长请求完成时间。现有工作改进了 MoE 通信、专家执行和服务组织 [1]–[4]。由于并发请求共享网络路径，带宽分配成为降低推理延迟的重要环节。

拥塞控制调节请求如何共享网络。数据中心传输协议使用显式拥塞通知、延迟或网内遥测来应对拥塞 [5]–[9]。要将网络观测转化为更好的请求性能，还需要了解每次传输所依赖的计算。两种分配方式都可以让同一瓶颈保持繁忙，但可能产生不同的 GPU 等待时间和请求完成时间。现在应为哪项未完成通信提供服务

以推进推理执行?

答案由两个特性决定。首先，消费者进度取决于相关传输。一个集合通信可能已有多个成员完成，但掉队参与者仍在阻滞所需输入。当加速某个队列对使该依赖关系更接近满足状态时，就会推进执行。Coflow 调度捕获相关流 [10]–[12] 之间共享的完成目标；GPU 依赖关系则进一步确定当前阻碍计算的未完成传输。

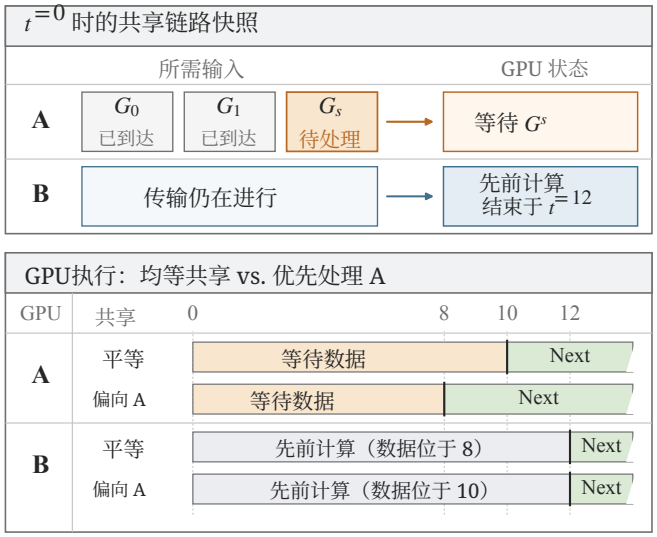

图 1. 通信服务与 GPU 进度。A 和 B 共享一条单位容量链路，剩余容量分别为 6 和 4 个单位。在两项传输均处于活动状态时，偏向 A 采用 3:1 的份额。A 在 $t = 8$ 而不是 10 时开始下一步计算；B 的输入稍后到达，但其下一步计算仍在 $t = 12$ 时开始。

其次，通信紧迫性会随着消费者进度而变化。已经完成前序工作的 GPU 在最后一个所需输入到达后即可启动。竞争中的另一个消费者可能仍有足以覆盖额外通信延迟的计算任务。将部分带宽转移给阻塞传输，可以推进第一个消费者，同时保持第二个消费者的启动时间不变（图 1）。集合通信依赖确定需要推进的工作；消费者就绪状态揭示其紧迫性以及可供重新分配的松弛量。

IACC 在 ACK/INT 反馈路径中融合这些信号。ACK/INT 观测决定共享有向链路上的有效服务预算，而运行时计算进度和依赖关系提示则指导集合通信之间的预算共享，以及每个集合通信内部的 QP 权重分配。具有计算余量的组会将服务让给消费者受阻的组，反馈路径则直接应用由此得到的绝对 QP 速率。

在每个连通分量中，IACC 选择一个一致的优先级组，使竞争链路推进同一依赖关系。受约束的渐进填充将组预算转换为符合路径和端点限制的速率。ACK/INT 观测会刷新共享约束；缓存速率

用于在刷新之间承载流量。

本文作出以下四项贡献：

- 我们解释了为什么流级服务可能偏离请求级进度：集合通信依赖关系确定哪些传输会阻塞执行，而消费者就绪状态决定它们的紧迫性。
- 我们设计了 IACC，在 ACK/INT反馈路径中结合网络遥测与推理状态，并直接应用由此产生的队列对速率。
- 我们分析了固定快照下的预算守恒和加权最大最小QP分配，包括算法终止和复杂度。
- 我们在基线和更高到达密度下评估了五种跟踪引导工作负载，测量 TTFT、每输出令牌时间、请求完成时间、GPU 等待和利用率。消费者状态消融实验和受控秩偏斜进一步考察了执行进度如何影响结果。在基线条件下，与 HPCC 相比，IACC 分别将平均 TTFT、每输出令牌时间和请求完成时间降低了 6.32%–19.41%、7.02%–27.33% 和 6.45%–21.47%；在通过 $1.25\times$ 的全部 30 种测试设置中，平均 TTFT 和第 95 百分位首令牌时间均更低。

II. 背景与动机

本节介绍 MoE 推理中的通信依赖关系及其与 GPU执行的交互。随后通过一个共享链路示例，说明需要同时考虑迟到传输和消费者进度的带宽分配方案。

A. *MoE* 推理中的通信与计算

混合专家（MoE）模型会为每个令牌选择一组专家。专家并行（EP）将专家分布在多个 GPU 上。MoE层将令牌分发给其分配的专家，执行专家计算，然后返回并合并结果。因此，通信连接了连续的计算阶段，并影响后续计算何时可以开始 [3], [4]。

每个 EP集合通信涉及多个秩，并通过多个队列对（QP）进行传输。输入相关路由会造成通信量和计算负载的差异，而并发集合通信可能会争用共享链路 [3]。它们的传输可能具有不同的路径和完成进度。同时，每个 GPU 的执行队列和依赖关系决定了其何时能够消费结果。

B. 从队列对进度到集体进展

RDMA拥塞控制利用网络反馈调整QP发送速率。例如，HPCC使用由确认包（ACK）携带的网内遥测（INT）来跟踪链路负载和队列状态 [9]。这些信号描述了网络状况；而集合通信的进展还取决于哪些必需传输仍未完成。

需要多个输入的消费者必须等所有输入到达后才能继续。因此，即使其他传输更早完成，某个迟到传输仍可能决定其通信依赖关系何时得到满足。共流研究同样识别出服务于同一应用的各流之间存在耦合完成目标应用阶段 [11]。集体成员关系和剩余工作量揭示了哪些传输限制了进度。它们的路径瓶颈同样至关重要：在一条路径上增加服务无法消除另一条路径上的通信尾部。

C. 从集体进展到*GPU*计算

考虑一项会消耗通信结果的GPU计算 G_i 。在给定的执行顺序下，设 R_i为除目标通信之外的所有先决条件均已满足的时间，设 F_i 为其所需传输完成的时间。最早开始时间和通信引起的等待时间为

$$S_i = \max(R_i, F_i), \tag{1}$$

$$W_i = \max(0, F_i - R_i). \tag{2}$$

当 $F_i > R_i$时，未完成通信会直接延迟计算，因此更早完成通信可以减少GPU 等待。当 $F_i \le R_i$时，通信被前置计算所掩盖；更早完成通信并不会使消费者提前开始。该前置计算不一定与目标通信同时开始。

剩余通信工作量决定所需的网络服务量。消费者就绪状态决定其紧迫性：等待数据的GPU需要及时获得服务，而正在进行的前置计算可能留出松弛量，从而允许延迟通信。

D. 基于依赖关系和进度的带宽分配

图 1 展示了共享有向瓶颈链路 e 的并发集合通信 A 和 B 。GPU G_c 上的计算 C_A 需要来自 G_0、G_1 和 G_s 的输入。前两个输入已到达，但来自 G_s 的传输尚未完成。由于 G_c 已完成其前置计算，它正在等待这最后一个输入。 B 的必要传输与 $G_s \to G_c$ 在 e 上竞争。不过，B 的消费者仍有前置计算正在进行；在当前分配下，其剩余通信可以在该计算结束前完成。

来自 G_0 和 G_1 的已完成输入无法解除 C_A 的阻塞：来自 G_s 的输入仍是阻塞依赖。因此，向 $G_s \to G_c$ 分配更多带宽可以缩短 GPU 等待。将 B 的部分带宽份额重新分配给这次传输，可以推进 C_A 的进度，同时允许 B 继续传输。尽管 B 较晚完成通信，只要传输仍能在前置计算结束前完成，其消费者就会在同一时间开始执行。可行的重新分配取决于 B 剩余的数据量、计算松弛量和路径容量。

两种分配都能充分利用链路，但会产生不同的 GPU 等待时间。集合通信依赖关系可以识别限制进度的传输；消费者状态则揭示了指导带宽分配的紧迫性和松弛量。

IACC 在 ACK/INT反馈路径中结合这些信号。运行时提供计算进度和依赖提示。这些提示与网络状态及剩余的集体通信工作量共同用于指导共享有向链路上的每组带宽预算

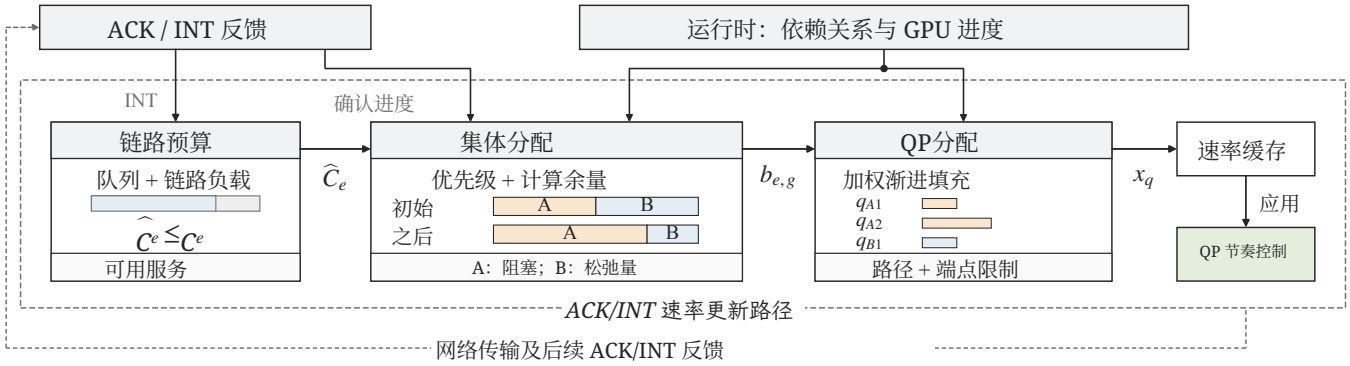

图 2. IACC 概览。ACK/INT 观测确定链路服务预算。集合通信依赖和 GPU 进度指导集体预算与 QP 权重；加权渐进填充强制执行分配约束。速率会被缓存，并在 ACK/INT 速率更新路径中直接应用。

以及这些预算向 QP 的分配，并直接更新其发送速率。下一节将介绍该设计。

III. IACC 设计

IACC 将网络反馈和推理进度转化为一个 QP 速率决策（图 2）。ACK/INT 路径首先确定每条有向链路上可用的服务量，然后利用集合通信依赖和消费者就绪状态进行分配，最后应用所得速率。运行时提供进度和依赖记录。我们将依次介绍分配目标、网络预算、集体分配和速率执行。

A. 模型与分配目标

在刷新时刻 t，令 $\mathcal{Q}$ 包含具有未发送数据和最新已交付记录的已接纳的 QP。QP q 属于集合通信$g(q)$，经过有向路径 $\mathcal{P}_q$，并具有端点上限 P_q。集合通信由请求和通信操作 ID 标识；不含集合通信元数据的流将形成自己的组。定义 $\mathcal{Q}_{e,g} = \{q \in \mathcal{Q} : g(q) = g, e \in \mathcal{P}_q\}$以及 $\mathcal{G}_e = \{g : \mathcal{Q}_{e,g} \neq \varnothing\}$。给定链路容量 C_e、基于反馈$\widehat{\ }$推导的预算 $C_e(t)$ 以及组预算 $\mathbf{b}$，可行的 QP 目标满足

$$0 \le x_q \le P_q, \qquad q \in \mathcal{Q}, \tag{3}$$

$$\sum_{q \in \mathcal{Q}_{e,g}} x_q \le b_{e,g}, \qquad \sum_{g \in \mathcal{G}_e} b_{e,g} \le \widehat{C}_e(t) \le C_e. \tag{4}$$

将该可行集记为 $\mathcal{X}(\mathbf{b})$。

令$F_{g,i}$ 为消费者 G_i所需通信的完成时间，令 $R_{g,i}$ 为排除该通信后的就绪状态。如第二节所述，通信引起的等待为 $W_{g,i} = [F_{g,i} - R_{g,i}]^+$。在保持其他依赖关系不变的情况下，将通信提前 $\epsilon \ge 0$ 可使等待时间减少

$$W_{g,i}(F_{g,i}) - W_{g,i}(F_{g,i} - \epsilon) = \min\{\epsilon, [F_{g,i} - R_{g,i}]^+\}. \tag{5}$$

此处为 $[z]^+ = \max(0,\ z)$。因此，当服务推进阻塞依赖时，便具有即时价值；被计算覆盖的通信则有松弛量。IACC 利用这一差异设定集合通信预算，然后在预算范围内计算队列对速率。

表 I IACC 使用的状态变量和决策变量。

符号	含义
u_q, o_q	QP q 的未发送和未确认字节
$\mathcal{B}_g$	当前因集合通信 g而阻塞的消费者
$R_{g,i},\ R_g$	消费者和最早的分组就绪时间
$L_{e,g},\ D_g$	剩余线上比特和估计排空时间
$\tilde{b}_{e,g}, b_{e,g}$	松弛量重新分配前后的预算
$x_q,\ \bar{x}_q$	连续目标和量化速率许可
$Q_e,\ \nu_e,\ H_e$	队列字节数、观测速率和排空时限
$C_e^{\mathrm{eff}}, \widehat{C}_e$	有效容量和可分配链路预算

B. 网络条件化链路预算

由确认携带的带内网络遥测提供队列字节数 Q_e、传输字节增量 ΔB_e和采样间隔 Δt_e。每个队列对的历史数据会汇入共享的有向边记录；第一个样本用于建立基线。要获得一个样本，队列对路径上必须存在唯一的边匹配，间隔必须为正，并且在时间范围$H_e = \max\{H_0, 4$往返时延$_q^{\mathrm{base}}\}$内有一条先前的确认。时间单位为秒、速率单位为比特/秒，定义

$$\nu_e = \frac{8\Delta B_e}{\Delta t_e}, \qquad u_e = \max\{1, \nu_e/C_e\}, \qquad z_e = \frac{8Q_e}{C_e H_e}. \tag{6}$$

负载项 u_e 和归一化排队工作量 z_e 决定有效容量和可分配预算：

$$C_e^{\mathrm{eff}} = \max\left\{r_{\min}, \left\lfloor \frac{C_e}{u_e + z_e} \right\rfloor\right\}, \qquad \widehat{C}_e = \eta C_e^{\mathrm{eff}}, \tag{7}$$

其中 $r_{\min} = 1$ bit/s，$0 < \eta \le 1$ 为利用率。边记录在 T_{obs} 后过期；缺失或过期观测将产生 $C_e^{\mathrm{eff}} = C_e$。观测生命周期和排空时限的用途不同：前者限制样本的存续时间，后者则用于归一化排队工作量。

对于 $C_e \ge r_{\min}$，分母确保 $C_e^{\mathrm{eff}} \le C_e$。在固定时间范围内，增加队列项或负载项不会增加预算。该预算会纳入排空估计及后续所有分配约束，使拥塞响应与可用于推理的服务能力相耦合。

算法1计算集体预算

输入： $\{C_e, L_{e,g}, V_g, R_g, \mathcal{B}_g\}$, t; 历史记录 $s,\ c$

使用 (10)–(11) 计算 D_g 和 σ_g
选择一致的优先级组
对于所有竞争分量 K **执行**
  $U_K \leftarrow \{g \in K :\ \mathcal{B}_g \neq \varnothing\}$
  如果 $c_K \notin K$ **则**
    $c_K \leftarrow$ 使 $_{g\in K}\, D_g$ 取最小值
  如果 $U_K \neq \varnothing$ **则**
    如果 $s_K \notin U_K$ **则**
      $s_K \leftarrow$ 使 $_{g\in U_K}\, \sigma_g$ 取最大值
    $h_K \leftarrow s_K$
  else
    $s_K \leftarrow \perp$; $h_K \leftarrow c_K$
分配和重新分配链路预算
对于所有 活动有向链路 e **执行**
  $K \leftarrow$ 包含 $\mathcal{G}_e$的组件
  $\mathcal{R}_e \leftarrow \{g :\ \exists q \in \mathcal{Q}_{e,\ g},\ \ q \sim \mathcal{B}_g\}$
  $a_{e,\ g} \leftarrow 1$ 对于所有 $g \in \mathcal{G}_e$
  如果 $(U_K = \varnothing$ 或 $\mathcal{R}_e \neq \varnothing)$ 且 $h_K \in \mathcal{G}_e$ **则**
    $a_{e,\ h_K} \leftarrow \alpha\ \ C_e a_{e,\ g} / \sum_h a_{e,}$
  $b_{e,\ g} \leftarrow \widetilde{b}_{e,g} \leftarrow \widehat{h}$
  if $\mathcal{R}_e = \varnothing$ **然后**
    继续
  $D_e \leftarrow 0$
  对于所有 $g \in \mathcal{G}_e \setminus \mathcal{R}_e$ **执行**
    $H_g \leftarrow R_g - t - \delta$
    if V_g 以及 $\mathcal{B}_g = \varnothing$ 以及 $D_g < H_g < +\infty$ **然后**
      $d \leftarrow \lceil \widetilde{b}_{e,g} -$ 最大值$(L_{e,g}/H_g, \rho b_{e,g})\rceil^+$
      $b_{e,\ g} \leftarrow b_{e,\ g} - d$; $D_e \leftarrow D_e + d$
  对于所有 $g \in \mathcal{R}_e$ **执行**
    $b_{e,\ g} \leftarrow b_{e,\ g} + D_e/|\mathcal{R}_e|$
返回 b

C. 集体协调与计算松弛

识别阻塞依赖。 运行时会报告集体成员关系、后继依赖、已提交的 GPU 工作以及通信进度。已交付记录提供每个消费者的就绪状态：

$$R_{g,i} = \max\{T_i^{\text{queue}}, T_{g,i}^{\text{own}}, T_{g,i}^{\text{release}}\}, \tag{8}$$

来自 GPU 队列尾部、前驱完成和操作释放。令 R_g 表示未完成消费者中的最早就绪状态。组有效性 V_g 检查成员关系、共同后继、最新 GPU 记录以及未完成数据证据。对于消费者 G_i，令 $\mathcal{I}_{g,\ i}$ 和 $\mathcal{O}_{g,\ i}$ 分别表示所需和观测到的关联流标识。阻塞集合 $\mathcal{B}_g$ 包含满足 $R_{g,\ i} \le t$、$\mathcal{O}_{g,\ i} = \mathcal{I}_{g,\ i}$ 的消费者，以及具有最新待处理传输记录和未完成通信的消费者。没有有效消费者元数据的组具有 $\mathcal{B}_g = \varnothing$，因此无法贡献计算余量。

令 u_q，o_q 表示未发送且未确认的字节数。计入分组载荷 m 和开销 h_{pkt}，其中 $h(v) = v + h_{\text{pkt}}\lceil v/m \rceil$，剩余线路工作量为

$$L_{e,g} = 8 \sum_{q \in \mathcal{Q}_{e,g}} h(u_q) + 8 \sum_{q \in \mathcal{F}_g :\, e \in \mathcal{P}_q} h(o_q), \tag{9}$$

算法 2加权渐进填充

输入： 约束行 A、容量 $\mathbf{c}$、权重 $\mathbf{w} > 0$

$\mathbf{x} \leftarrow \mathbf{0}$; $U \leftarrow \mathcal{Q}$
当 $U \neq \varnothing$ **时**
  $s_j \leftarrow \sum_{q\in U} A_{jq} w_q$ 对于每一行 j
  $\Delta \leftarrow$ 最小$_{j:s_j>0}\{[c_j - (A\mathbf{x})_j]/s_j\}$
  对于所有 $q \in U$, **执行**
    $x_q \leftarrow x_q + w_q \Delta$
  $J \leftarrow \{j : (A\mathbf{x})_j = c_j\}$
  $U \leftarrow U \setminus \{q :\ \exists j \in J,\ \ A_{jq} = 1\}$
返回 x

其中 $\mathcal{F}_g$ 包括正在等待最终确认的已观测流。未完成确认的字节在收到确认之前仍计入其路径。

协调集合优先级。 均分预算提供了一个参考排空估计：

$$b_{e,g}^0 = \frac{\widehat{C}_e}{|\mathcal{G}_e|}, \qquad D_g = \max_{e: g \in \mathcal{G}_e} \frac{L_{e,g}}{b_{e,g}^0}. \tag{10}$$

对于存在受阻消费者的组，优先级评分

$$\sigma_g = \frac{|\mathcal{B}_g|}{\max(\tau_0, D_g)} \tag{11}$$

衡量每单位估计排空时间内的等待中的消费者数量；τ_0 在接近完成时限制该评分。 D_g中的最大值捕获最受限制的路径段，因此该评分同时反映剩余的集体任务以及它能够释放的消费者。

连接共享有向链路的集合通信。在每个连通分量 K 内，受阻消费者的获胜者 s_K在符合条件时保持优先级；否则由最大的 σ_g 获胜。独立地，连续性组 c_K 在处于活动状态时持续存在，否则由最小的 D_g所在组替代。当不存在受阻组时，它提供优先级。平局时使用集体 ID。一个集合通信可以沿其路径与不同的组竞争。独立的链路优先级可能会将服务分散到相互竞争的依赖关系中。组件范围的获胜者为这些链路提供共同方向，同时每个预算仍特定于遍历该链路的组。

当 q 的任一端点属于 $\mathcal{B}_g$时，写入 $q \sim \mathcal{B}_g$，并让 $\mathcal{R}_e$ 包含在 e上具有此类队列对的组。存在受阻消费者时，获胜者在满足 $\mathcal{R}_e \neq \varnothing$的链路上获得权重 $\alpha \ge 1$；其他链路保留均分。不存在受阻组时，连续性获胜者沿其路径获得该权重。初始预算为

$$\widetilde{b}_{e,g} = \widehat{C}_e \frac{a_{e,g}}{\sum_{h \in \mathcal{G}_e} a_{e,h}}, \qquad a_{e,g} \in \{1, \alpha\}. \tag{12}$$

重新分配计算余量。 捐赠者具有有效状态、没有受阻消费者、有限的 R_g，并且在其最早的消费者就绪之前有足够的时间：

$$H_g = R_g - t - \delta > D_g, \tag{13}$$

其中 δ 预留了一个保护间隔。使用最早尚未完成的消费者：当组中的任一消费者首次需要通信时，窗口即结束

算法 3联合 ACK/INT 速率更新

输入： ACK/INT a，缓存 Λ，上次刷新时间 t_0

▷**事件：收到队列对 q 的确认时：**
$t \leftarrow$ Now(); RecordProgress(q, a)
$\mathcal{N} \leftarrow$ 更新网络($q, a.I, t$)
如果缓存未初始化或 $t - t_0 \geq \Delta_{\text{ref}}$ **则**
  $S \leftarrow$ReadDelivered(t)
  $\widehat{\mathbf{C}} \leftarrow$ 链接预算($\mathcal{N}, t$) # Eq. 7
  (V, R, $\mathcal{B}$) $\leftarrow$ 分类(S, t)
  $\mathbf{b} \leftarrow$ 分组预算(S, $\widehat{\mathbf{C}}$ $V, R, \mathcal{B}, t$)
  (A,$\mathbf{c}$,$\mathbf{w}$) $\leftarrow$ 构建约束(S,$\mathbf{b}$ $\mathcal{B}$)
  $\mathbf{x} \leftarrow$ 渐进填充($A, \mathbf{c}$,$\mathbf{w}$) # Alg. 2
  $\Lambda \leftarrow \varnothing$
  对所有 $j \in \mathcal{Q}$ **执行**
    $\Lambda[j] \leftarrow$ (最大值($r_{\min}$, $\lfloor x_j \rfloor$) $t + T_{\text{life}}$)
  $t_0 \leftarrow t$; $J \leftarrow$ 活动管理的查询处理程序()
否则
  $J \leftarrow \{q\}$
对于所有 未完成的 $j \in J$ **执行**
如果 $j \notin \Lambda$ 或 $t > \Lambda[j]$.到期 **则**
  继续 # 保持已应用速率
  应用速率控制(j，最大{$r_{\min}$,最小($\Lambda[j]$.速率 P_j)})

。其保留预算和捐赠为

$$k_{e,g} = \max\{L_{e,g}/H_g, \rho\widetilde{b}_{e,g}\}, \tag{14}$$

$$d_{e,g} = [\widetilde{b}_{e,g} - k_{e,g}]^+, \qquad 0 < \rho \leq 1. \tag{15}$$

$k_{e,\,g}$ 中的各项覆盖可用时域内已观测的工作量，并保持服务下限。算法1汇总每条链路上的捐赠量，并在 $\mathcal{R}_e$之间平等分配。时域条件规定了初始优先级分配后的重新分配。接收方不必是组件赢家：资格取决于该链路所服务的受阻消费者。优先级协调组件间的集合通信进展，而松弛量重新分配利用计算重叠来协助阻塞传输。

命题1（预算守恒）。在连续算术中，算法1满足$\sum_g b_{e,\,g} = C_e \leq C_e$ 和$\sim b_{e,\,g} \geq \rho b_{e,\,g} > 0$。证明。初始份额之和为 C_e。捐赠方$\tilde{}$最多释放$(1-\rho)b_{e,\,g}$，而接收方恰好收到所释放份额之和。向下舍入保持容量不等式。□

*D. QP*分配和速率执行

在集体预算内进行分配。QP权重将剩余工作量与受阻消费者优先级相结合：

$$w_q = \begin{cases} \max(1, u_q), & q \sim \mathcal{B}_{g(q)}, \\ \max(1, u_q)/\beta, & \text{otherwise}, \end{cases} \qquad \beta \geq 1. \tag{16}$$

对于分类相同的QP，比例速率会使剩余序列化时间趋于相等，直到某个路径约束生效；β 优先考虑与受阻消费者关联的QP。

将链路–组和端点约束分别表示为 $A\mathbf{x} \leq \mathbf{c}$。链路–组行选择 $\mathcal{Q}_{e,\,g}$ 并具有容量 $b_{e,\,g}$；而

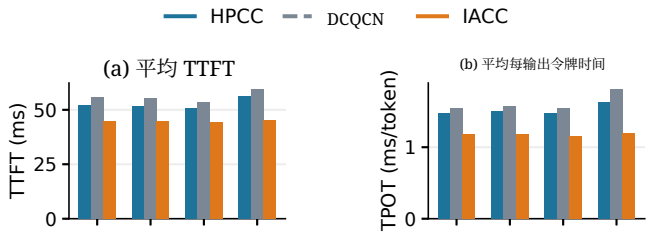

图3。 1.0×到达密度下的平均 TTFT 和每请求每输出令牌时间。图中展示了四种工作负载；BurstGPT 也纳入到达密度扫描中，并汇总统计数据。数值越低越好。

端点行选择一个 QP，并具有容量 P_q。IACC 应用加权渐进填充以获得

$$\operatorname*{lexmax}_{\mathbf{x} \in \mathcal{X}(\mathbf{b})} \operatorname{sort}_{\uparrow}\{x_q/w_q : q \in \mathcal{Q}\}. \tag{17}$$

对于活跃QP U，定义$s_j = \sum_{q \in U} A_{jq} w_q$。每个填充步骤都以斜率 w_q 增加它们的速率，增量为可行的最大公共增量：

$$\Delta = \min_{j : s_j > 0} \frac{c_j - (A\mathbf{x})_j}{s_j}. \tag{18}$$

算法2冻结触及饱和约束的QP，并继续处理其余QP。

命题2（固定快照分配）。对于固定预算和正权重，算法 2 至多经过 $n = |\mathcal{Q}|$ 次迭代即可返回一个可行的加权最大最小分配。证明。公式（18）保持可行性，并且每一步至少冻结一个 QP。每个被冻结的 QP 都达到端点上限，或与归一化速率不高于自身的 QP 共享一个饱和预算，从而建立加权最大最小公平性。□

借助 $M = \sum_q |\mathcal{P}_q|$，稀疏扫描的时间复杂度为 $O(n(n+M))$，存储开销为 $O(n+M)$。

应用速率决策。算法 3 在每次确认时更新网络和进度记录。刷新时，它会计算链路预算、集体预算和 QP 速率，替换缓存 Λ，并更新匹配的未完成 QP。在两次刷新之间，仅考虑当前确认对应的 QP。对于量化目标 $\bar{x}_q = \max\{r_{\min}, \lfloor x_q \rfloor\}$，执行过程如下

$$r_q(t^+) = \begin{cases} \max\{r_{\min}, \min(\bar{x}_q, P_q)\}, & \Lambda[q] \text{ valid}, \\ r_q(t^-), & \text{otherwise}. \end{cases} \tag{19}$$

新的 QP 从 P_q 开始；缺失或已过期的授权会保留已应用速率。因此，网络和消费者状态共同决定一个直接应用的绝对速率。量化带来的增量最多为$n_e r_{\min}$，高于连续链路预算，其中 n_e 统计 e上的受管理 QP 数量。分配属性适用于每个固定快照；观测延迟、缓存以及传输中的数据包也会影响执行。

IV. 评估

我们评估 IACC 对令牌延迟和 GPU 等待的影响。首先，我们比较五种

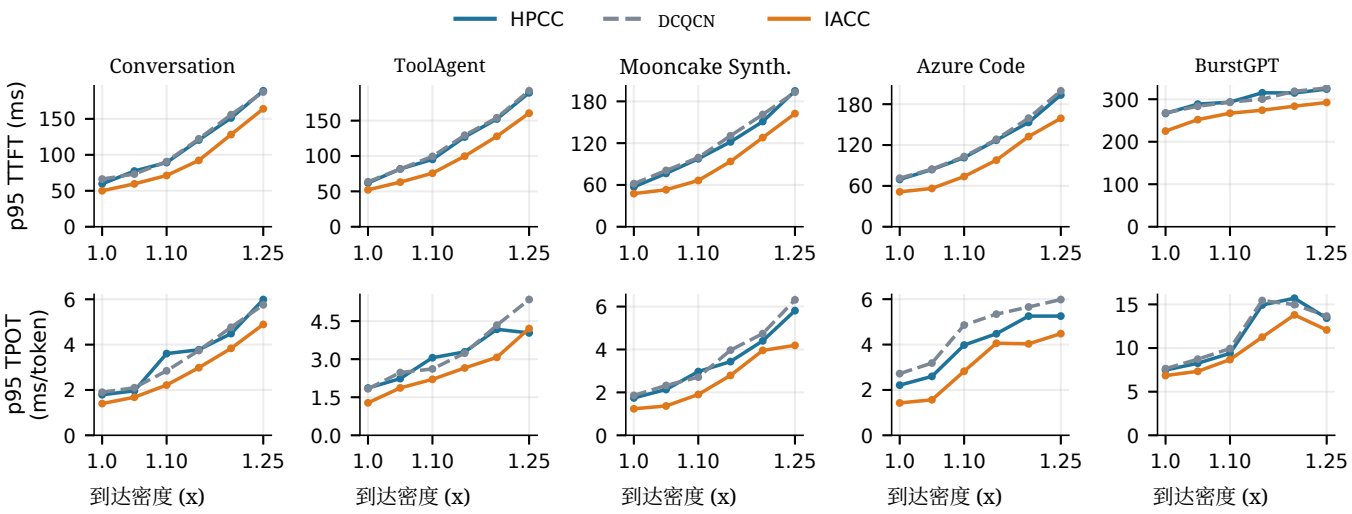

图 4. 第 95 百分位首令牌时间（上）和每请求第 95 百分位每输出令牌时间（下）随到达密度的变化。每列对应一种工作负载；各面板的纵轴尺度不同。在每种设置下，各方法使用相同的输入和测量窗口。

工作负载的端到端性能，然后将收益与通信和计算进度联系起来。最后，我们分别分析消费者状态信息和集合内 QP 加权的影响。

A. 实验设置

仿真环境。 我们使用 SimAI 的数据包级 ns-3 后端 [13] 评估 IACC。集群包含 16 个 GPU，采用解耦式预填充/解码部署：12 个预填充 GPU 构成三个 EP 组，四个解码 GPU 构成一个 EP 组，所有 GPU 均配置 EP=4。四台叶交换机连接到一台脊交换机，每台叶交换机连接四个 GPU。每个 EP 组都跨越全部四台叶交换机，因此并发集合通信和键值传输之间会产生争用。GPU 与叶交换机之间、叶交换机与脊交换机之间的链路带宽分别为 50 Gb/s 和 25 Gb/s，每条链路的传播延迟为 $2\,\mu$。[1]

工作负载。 我们使用五种轨迹引导的推理工作负载：Conversation、ToolAgent，以及来自 Mooncake [14]、Azure Code [15]，和 BurstGPT [16] 的 Mooncake Synthetic。轨迹窗口提供到达模式以及输入/输出长度顺序，并将其映射为合成预填充/解码模板，其中明确包含计算和通信依赖关系。每个请求包含 8、12 或 16 次解码迭代。所选轨迹窗口未经压缩的到达序列定义了 $1.0\times$ 到达密度。到达密度从 $1.0\times$ 变化至 $1.25\times$，步长为 $0.05\times$，并在 $1.5\times$、$1.75\times$ 和 $2.0\times$ 处进行额外测试。提高到达密度会压缩请求到达间隔，同时保持请求顺序和每个请求的工作量不变。所有流量均来自推理工作负载。

比较与测量。 我们将 IACC 与 HPCC [9]和 DCQCN [6]进行比较。各方法在相同的拓扑上使用完全相同的请求输入、放置方式和执行依赖关系。每种工作负载–密度设置使用一个跟踪窗口，并为每种方法进行一次配对运行。运行包括 60 秒的仿真预热阶段和 600 秒的测量窗口，随后进行排空。统计数据涵盖测量窗口期间到达的所有请求，包括在窗口结束后完成的请求。

指标。 我们的主要指标是平均首令牌时间（TTFT）和第 95 百分位首令牌时间（TTFT），以及平均每输出令牌时间（TPOT）和第 95 百分位每输出令牌时间（TPOT）。对于请求到达时间 a_r 和令牌就绪时间 $t_{r,1}$、……、t_{r,N_r}，TTFT 为 $t_{r,1}-a_r$，TPOT 为 $(t_{r,N_r}-t_{r,1})/(N_r-1)$。TPOT 在聚合前按请求计算。当某次解码迭代所需的通信和计算完成时，令牌即视为就绪。我们还报告请求完成时间（RCT）、$t_{r,N_r}-a_r$，以及同时满足 TTFT≤ 55毫秒和 TPOT≤ 1.7毫秒/令牌的请求比例。

为解释这些性能提升，我们测量集体完成时间、KV 交接延迟、GPU 等待和利用率。集合通信完成时间从首次发起持续到最后一个排名完成；KV 交接从生产者就绪状态持续到所有所需数据到达。GPU 等待表示后续计算仍在进行期间的空闲时间。对于每个 GPU，任务区间在预填充的请求到达时刻或解码的完整 KV 到达时刻开始，并在最后一次持续时间为正的计算结束时结束。重叠区间会被合并。利用率是这些区间内用于计算的时间比例，并在各 GPU 之间取平均。

B. 端到端性能

不同工作负载下的令牌延迟。图3比较了在 $1.0\times$ 到达密度下的 Conversation、ToolAgent、Mooncake Synthetic 和 Azure Code。在包括 BurstGPT 在内的全部五种工作负载中，与 HPCC 相比，IACC 将平均 TTFT 降低了 6.32%–19.41%，将平均每输出令牌时间降低了 7.02%–27.33%，并将平均请求完成时间降低了 6.45%–21.47%。AzureCode 的平均 TTFT 和平均每输出令牌时间降幅最大，而突发性较强的 BurstGPT 时间窗口则改善较小。

[1]在内部，模拟器将时间按 $1000\times$ 进行缩放，并以相同的倍数降低链路速率。报告的延迟采用参考尺度；时间序列坐标轴保留模拟时间。

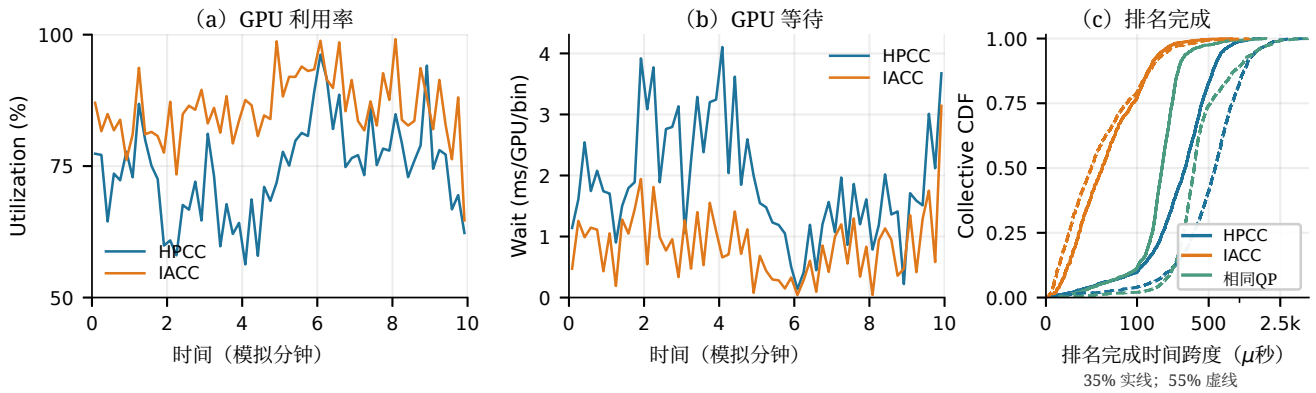

图5. Conversation 在 1.0×时的通信与 GPU 进度。（a）–（b）原始工作负载：以 10 个模拟秒为分箱统计 GPU 利用率和 GPU 等待。利用率采用任务间隔内 busy/(busy+waiting) 的计算方式。（c）受控秩不平衡：实线/虚线累积分布函数分别表示 35%/55% 的热点对占比。扩散轴在 100 μs 以内为线性，之后为对数刻度。

递增的到达压力。图4展示了到达密度增加时第 95 百分位首令牌时间和每输出令牌时间的变化。与 HPCC 相比，在全部 30 个工作负载—密度设置中，IACC 均降低了平均 TTFT 和第 95 百分位首令牌时间，降幅分别为 5.81%–28.02% 和 8.85%–33.00%。平均每输出令牌时间在 27 个设置中有所改善，第 95 百分位每输出令牌时间在 29 个设置中有所改善；平均请求完成时间则降低了 6.45%–30.11%。对于 1.25× 下的 ToolAgent，第 95 百分位每输出令牌时间比 HPCC 高 4.25%。总体而言，随着到达压力增加，IACC 在大多数设置下改善了令牌延迟，并缩短了请求完成时间。

联合服务质量。我们对每种设置均采用相同的首令牌时间和每输出令牌时间预算。在 1.0×，Conversation、ToolAgent、Mooncake Synthetic 和 Azure Code 的所有 IACC 请求均满足这两项预算，而 HPCC 下仅为 42.86%–75.00%。在 1.10×，IACC 在这些工作负载上的联合通过率达到 38.71%–67.74%，而 HPCC 仅为 0%–9.68%。在基线密度下，BurstGPT 在三种方法中的联合通过率均为零。在 1.5×及以上的密度下，五种工作负载的联合通过率均为零。在 2.0×，HPCC 的测量队列请求中有 75.44%–84.62%、IACC 的请求中有 71.93%–84.62% 在窗口边界处仍未完成。这些请求会在排空期间完成，并仍计入延迟统计。结果反映的是测量窗口内高负载下的性能，而不是最大可持续到达率。

C. 理解性能提升

从通信进度到令牌延迟。我们使用 1.0× 中的 Conversation 来考察总体性能提升背后的各个阶段。在这一示例中，全部 28 个匹配的请求 ID 在 IACC 下的 TTFT、TPOT 和 RCT 均更低。图5(a)–(b)跟踪了整个共同测量窗口内的 GPU 利用率和 GPU 等待情况。

表 II 将平均 TTFT 分解为生产者处从到达到键值就绪、键值交接以及从键值到达到首令牌这三个阶段。IACC 缩短了这三个组件的耗时。预填充和解码的集合通信完成时间分别下降了 19.88% 和 21.55%，而键值交接耗时下降了 13.90%。从到达到预填充就绪的时间间隔包含排队、计算和预填充通信，将通信变化与整个执行路径的进度联系起来。

表 II 1.0×中 Conversation 的平均 TTFT 分解（毫秒）。预填充包括到达至生产者 KV 就绪；KV 后处理在首个令牌处结束。

方法	预填充	KV 后处理	TTFT
HPCC	43.849	7.224	1.161 52.235
IACC	38.072	6.220	0.648 44.940

减少空闲间隔。GPU 等待时间下降了 54.86%，利用率则从 73.08% 提升至 85.65%。因此，更低的令牌延迟伴随着主动执行期间更少的空闲时间，而观测窗口内的计算量几乎未变。这将请求级增益与通信–计算依赖链上更短的等待时间联系起来。

受控秩不平衡。令牌路由会在各秩之间产生不均衡的专家需求和全对全流量。Lina 报告称，在其包含四个和十六个专家的推理任务中，最受欢迎的专家接收的令牌数量分别是最不受欢迎专家的 4.02 倍和 5.56 倍 [3]。Tutel 还识别出动态变化的专家负载 [17]。我们采用受控流量偏斜，以考察不均衡的秩进度如何影响消费者等待和令牌延迟。

对于 1.0× 中的 Conversation，我们保留请求到达、计算、键值流量以及每个集合通信的总有效载荷，仅改变秩对之间的分布。EP=4会产生 12 条有向流。一个热点秩对的两条互为反向的流分别承载有效载荷的 35% 或 55%；其余 10 条流平分剩余部分。热点秩对会在通信轮次之间轮换。均匀分布会将 16.7% 分配给一个秩对，因此这两种设置分别将份额集中到均匀份额的 2.1 倍和 3.3 倍。表 III 还包括 NG 和 EQ，其定义见第 IV-D 节。

图 5(c) 展示了秩完成时间差，即最早与最晚

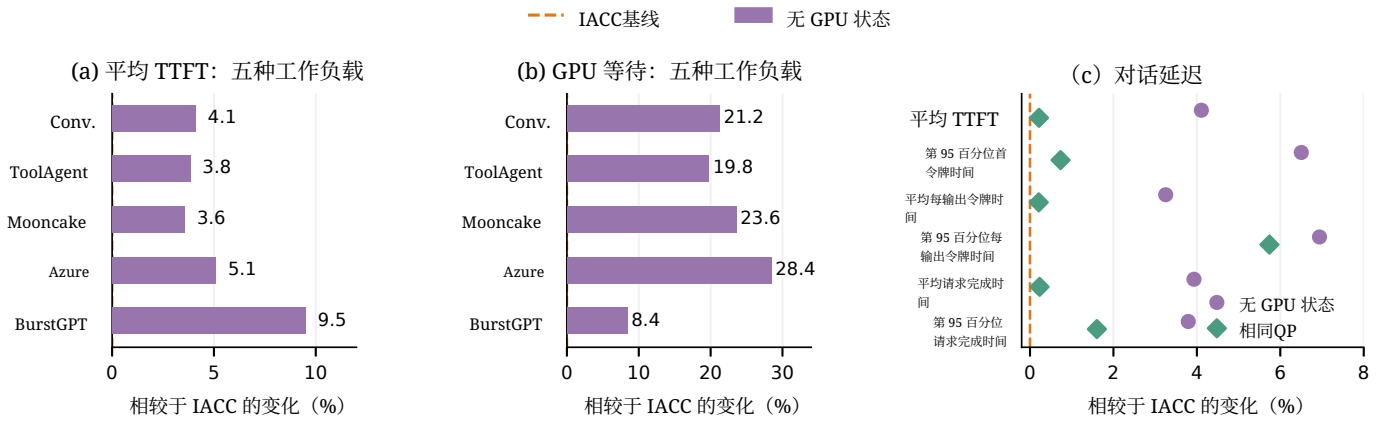

图 6. 在 1.0× 到达密度下的消融实验。(a)–(b) 在五种工作负载中禁用消费者状态后，平均 TTFT 和 GPU 等待的变化。(c) NG 和 EQ 下 Conversation 的平均和第 95 百分位请求延迟。数值表示相对于完整 IACC 的百分比变化；虚线零线表示完整 IACC。正值表示延迟或等待时间增加。

表 III 受控流量偏斜下的延迟和秩完成时间差。TTFT 单位为毫秒，TPOT 单位为毫秒/令牌，时间差单位为 μ秒。除标记为第 95 百分位的列外，其余列报告均值。

份额	方法	首令牌时间	每输出令牌时间	第 95 百分位每输出令牌时间	秩完成时间差
35%	HPCC	96.00	2.448	3.515	325.8
	IACC	81.65	2.078	2.737	75.3
	NG	112.59	1.743	2.348	46.5
	EQ	76.03	2.043	3.323	193.0
55%	HPCC	134.32	3.515	4.903	659.1
	IACC	127.00	2.986	4.502	72.8
	NG	224.71	3.627	5.206	51.6
	EQ	129.48	3.172	4.667	511.2

秩完成时间之间的差值。我们还通过合并最早秩完成后、消费者仍在等待通信期间的 GPU 空闲区间，测量本地消费者等待时间。在 35% 偏斜下，IACC 将平均秩完成时间差从 325.8 降至 75.3 μs；在 55% 偏斜下，则从 659.1 降至 72.8 μs。本地消费者等待时间分别从 66.86 降至 15.04ms/GPU，以及从 113.54 降至 12.21 ms/GPU。

表 III 将秩进度与端到端延迟联系起来。在 35% 偏斜下，与 HPCC 相比，IACC 将平均 TTFT、每输出令牌时间和请求完成时间分别降低了 14.95%、15.10% 和 14.69%；其第95百分位值分别下降了 16.69%、22.13% 和 18.22%。在 55% 偏斜下，平均 TTFT 和每输出令牌时间分别降低了 5.45% 和 15.06%。这些结果表明，更集中的排名完成和更短的本地等待能够改善令牌延迟。

相较于 EQ，在 35% 偏斜下，完整 IACC 将第 95 百分位每输出令牌时间降低了 17.63%，将平均秩完成时间差降低了 60.97%，并将本地消费者等待降低了 58.45%。平均首令牌时间、平均每输出令牌时间和 GPU 等待总时长分别增加了 7.40%、1.71% 和 9.82%。在 55% 偏斜下，平均首令牌时间和每输出令牌时间分别下降了 1.91% 和 5.86%。因此，组内加权的影响取决于流量分布和执行阶段暴露于通信延迟。

D. 消融实验

我们分别考察消费者状态感知和组内加权。所有方法均使用相同的请求输入、计算依赖关系和网络配置，仅禁用相应的组件。

消费者状态感知。无 *GPU* 状态（NG）会移除 GPU 繁忙状态、队列和消费者紧迫性信息，同时保留集合通信进度和链路约束。图6(a)– (b) 比较了 1.0× 到达密度下的五种工作负载。禁用消费者状态后，平均 TTFT 增加了 3.57%–9.51%，GPU 等待增加了 8.39%–28.42%。消费者信息有助于速率决策识别限制计算的通信，从而减少等待和请求延迟。

组内加权。 相同*QP* (EQ) 保留消费者状态、组预算和链路约束，但将所有组内 QP 权重设为 1。这样可以单独考察非均匀加权的影响。在五种原始工作负载中，与完整 IACC 相比，EQ 使平均 TTFT 变化 -0.04% 至 $+0.24\%$，使平均每输出令牌时间变化 -6.56% 至 $+0.89\%$。其对平均延迟的影响较小，且取决于工作负载。

图6(c) 比较了 Conversation 在 1.0×时的六项请求延迟指标。与这两种变体相比，完整 IACC 的六项延迟指标值均更低。与 NG 相比，平均 TTFT、每输出令牌时间和请求完成时间分别下降 3.95%、3.15% 和 3.78%。与 EQ 相比，三项均值分别下降 0.22%、0.21% 和 0.23%，其第95百分位值分别下降 0.73%、5.43% 和 1.58%。在此工作负载中，加权的主要优势体现在每输出令牌时间的长尾表现上。

在这种情况下，完整 IACC 下的 GPU 等待为 50.88 ms/GPU，略高于 EQ 的 49.92ms/GPU；两者的利用率分别为 85.65% 和 85.88%。结合第IV-C节中的负载不均衡实验，这些结果更清楚地表明，消费者状态对平均请求延迟有何影响。组内

加权会改变通信完成顺序及其对后续计算的影响。

V. 相关工作

数据中心拥塞控制。数据中心拥塞控制方案采用不同的信号和目标。DCTCP 和 DCQCN 使用 ECN反馈，而 TIMELY 和 Swift 则根据延迟推断拥塞 [5]–[8]。HPCC 和 PowerTCP 利用网内遥测估算链路利用率并对拥塞作出响应 [9], [18]；PCC 根据观测到的端到端性能选择速率调整策略 [19]。其他方案则着眼于延迟或流完成：HULL 预留带宽余量以限制排队 [20], , pFabric 根据流的完成要求为流设置优先级 [21], , D^3 则依据流截止期限分配速率 [22]。这些方案共同确立了反馈驱动的速率控制和流级优先级调度。IACC 在此基础上，将集合通信依赖关系和 GPU 就绪状态纳入考虑，以便在竞争的队列对之间分配服务。

应用感知带宽分配。D^2TCP 将流截止期限纳入拥塞控制 [23]。共流研究将相关流的集合表示为具有共同应用级完成目标的整体 [10]；Varys 利用这一抽象调度通信阶段，而 Aalo 则设计了非预知式Coflow调度器 [11], [12]。MLTCP 利用迭代进度协调分布式训练作业之间的通信 [24], ，而 ByteScheduler 在保留框架依赖关系的同时调度并划分 DNN训练传输 [25]。这些系统表明，应用结构能够指导网络服务。IACC 着重关注推理时消费者就绪状态：它识别阻塞 GPU执行的通信，以及能够吸收延迟交付的计算间隔，随后利用这一状态，在 ACK/INT反馈路径内的集合通信之间重新分配带宽。

MoE 通信优化。MoE系统优化专家放置、负载均衡和通信重叠。DeepSpeed-MoE 和 Tutel 提供可扩展的 MoE 执行能力，而 FasterMoE 和 Lina 致力于解决分布式 MoE 工作负载中的通信和专家负载低效问题 [1]–[3], [17]。MegaBlocks 改进了稀疏专家计算，MegaScale-Infer 则通过解耦和流水线处理支持 MoE 服务 [4], [26]。这些系统在模型或服务系统层面组织计算与通信。IACC 关注一个互补的传输问题：给定并发集合通信及其运行时依赖关系，应如何划分共享链路服务，以推进等待通信结果的 GPU 计算?

大语言模型服务系统。服务系统还会调度工作负载，以平衡吞吐量和延迟。Orca 针对自回归生成引入了按迭代调度，Sarathi- Serve 则采用分块预填充，以减少预填充与解码迭代之间的相互干扰 [27], [28]。DistServe 和 Splitwise 将预填充与解码分配到不同的 GPU 或机器上，以管理二者不同的资源需求 [29], [30]。这些系统主要调整请求批处理、阶段放置或GPU分配。相比之下，IACC 根据网络反馈以及 GPU消费者的就绪状态，控制并发通信流的速率，并且可以与这些服务层策略互为补充。

VI. 结论

分布式 MoE 推理将网络服务与集合通信依赖关系及 GPU 就绪状态相结合，因此仅凭链路利用率无法衡量通信对请求进度的影响。IACC 将这些执行信号纳入基于 ACK/INT 的 QP 速率分配；其固定快照分析证明了预算守恒和加权最大最小分配。在五种跟踪引导工作负载中，相较于基线到达密度下的 HPCC，IACC 将平均 TTFT、每输出令牌时间和 RCT 分别降低了 6.32%–19.41%、7.02%–27.33% 和 6.45%–21.47%。在到达密度不超过 $1.25\times$ 的情况下，所有 30 种测试设置中的平均 TTFT 和第 95 百分位首令牌时间均有所降低。对于 Conversation，GPU 等待降低了 54.86%，而利用率则从 73.08% 提升至 85.65%。这些结果表明，在所评估的工作负载中，使网络服务与消费者就绪状态相匹配能够改善端到端进度。

参考文献

[1] S. Rajbhandari、C. Li、Z. Yao、M. Zhang、R. Y. Aminabadi、A. A. Awan、J. Rasley 和 Y. He，“DeepSpeed-MoE：推进混合专家推理与训练，以支持下一代 AI 规模”，载于 第 *39* 届国际机器学习会议论文集，机器学习研究论文集，第 162 卷，2022 年，第 18 332–18 346 页。 [在线]。可访问：https://proceedings.mlr.press/v162/rajbhandari22a.html [2] J. He、J. Zhai、T. Antunes、H. Wang、F. Luo、S. Shi 和 Q. Li，“Faster-MoE：大规模动态预训练模型训练的建模与优化”，载于 第 *27* 届 *ACM SIGPLAN* 并行编程原理与实践研讨会（*PPoPP 22*）论文集，2022 年，第 120–134 页。[3] J. Li、Y. Jiang、Y. Zhu、C. Wang 和 H. Xu，“借助 Lina 加速分布式 MoE 训练与推理”，载于 *2023 USENIX* 年度技术会议（*ATC 23*），2023 年。 [在线]。可访问：https://www.usenix.org/conference/atc23/presentation/li-jiamin [4] R. Zhu、Z. Jiang、C. Jin、P. Wu、C. A. Stuardo、D. Wang、X. Zhang、H. Zhou、H. Wei、Y. Cheng、J. Xiao、X. Zhang、L. Liu、H. Lin、L.-W. Chang、J. Ye、X. Yu、X. Liu、X. Jin 和 X. Liu，“MegaScale-Infer：采用解耦专家并行的高效混合专家模型服务”，载于 *ACM SIGCOMM 2025* 论文集，2025 年。[5] M. Alizadeh、A. Greenberg、D. A. Maltz、J. Padhye、P. Patel、B. Prabhakar、S. Sengupta 和 M. Sridharan，“数据中心 TCP（DCTCP）”，载于*ACM SIGCOMM 2010* 会议论文集：计算机通信的应用、技术、架构与协议，2010 年，第 63–74 页。[6] Y. Zhu、H. Eran、D. Firestone、C. Guo、M. Lipshteyn、Y. Liron、J. Padhye、S. Raindel、M. H. Yahia 和 M. Zhang，“大规模 RDMA 部署的拥塞控制”，载于*2015* 年 *ACM* 数据通信特别兴趣小组会议论文集，2015 年，第 523–536 页。[7] R. Mittal、V. T. Lam、N. Dukkipati、E. Blem、H. Wassel、M. Ghobadi、A. Vahdat、Y. Wang、D. Wetherall 和 D. Zats，“TIMELY：基于往返时延的拥塞控制”，载于*2015* 年 *ACM* 数据通信特别兴趣小组会议论文集，2015 年，第 537–550 页。[8] G. Kumar、N. Dukkipati、K. Jang、H. M. G. Wassel、X. Wu、B. Montazeri、Y. Wang、K. Springborn、C. Alfeld、M. Ryan、D. Wetherall 和 A. Vahdat，“Swift：延迟是数据中心拥塞控制的简单而有效的方法”，载于 *ACM SIGCOMM 2020* 论文集，2020 年，第 514–528 页。[9] Y. Li、R. Miao、H. H. Liu、Y. Zhuang、F. Feng、L. Tang、Z. Cao、M. Zhang、F. Kelly、M. Alizadeh 和 M. Yu，“HPCC：高精度拥塞控制”，载于*ACM* 数据通信特别兴趣小组会议论文集，2019 年，第 44–58 页。

[10] M. Chowdhury 和 I. Stoica，“Coflow：面向集群应用的网络抽象”，载于第 11 届 ACM 网络热点研讨会（HotNets-XI）论文集，2012 年，第 31–36 页。[11] M. Chowdhury、Y. Zhong 和 I. Stoica，“借助 Varys 实现高效的 Coflow 调度”，载于 ACM SIGCOMM 2014 论文集，2014 年。[12] M. Chowdhury 和 I. Stoica，“无需先验知识的高效 Coflow 调度”，载于 2015 年 ACM 数据通信特别兴趣小组会议论文集（SIGCOMM 15），2015 年，第 393–406 页。[13] X. Wang、Q. Li、Y. Xu、G. Lu、D. Li、L. Chen、H. Zhou、L. Zheng、S. Zhang、Y. Zhu、Y. Liu、P. Zhang、K. Qian、K. He、J. Gao、E. Zhai、D. Cai 和 B. Fu，“SimAI：通过可扩展性与精度统一大规模大语言模型训练的架构设计与性能调优”，载于第 22 届 USENIX 网络系统设计与实现研讨会（NSDI 25），2025 年，第 541–558 页。[14] Mooncake 贡献者，“Mooncake FAST ’25 工作负载跟踪数据”，2025 年。[在线]。可访问：https://github.com/kvcache-ai/Mooncake/tree/main/FAST25-release/traces[15] Microsoft，“Azure LLM 推理数据集 2024”，2024 年。[在线]。可访问：https://github.com/Azure/AzurePublicDataset/blob/master/AzureLLMInferenceDataset2024.md[16] 高性能机器学习实验室，“BurstGPT：面向大语言模型服务的真实世界工作负载数据集”，2024 年。 [在线]。可访问：https://github.com/HPMLL/BurstGPT [17] C. Hwang、W. Cui、Y. Xiong、Z. Yang、Z. Liu、H. Hu、Z. Wang、R. Salas、J. Jose、P. Ram、H. Chau、P. Cheng、F. Yang、M. Yang 和 Y. Xiong，“Tutel：大规模自适应混合专家”，载于机器学习与系统论文集，第 5 卷，2023 年，第 269–287 页。[18] V. Addanki、O. Michel 和 S. Schmid，“PowerTCP：突破数据中心网络的性能极限”，载于第 19 届 USENIX 网络系统设计与实现研讨会（NSDI 22），2022 年，第 51–70 页。 [在线]。可访问：https://www.usenix.org/conference/nsdi22/ presentation/addanki [19] M. Dong、Q. Li、D. Zarchy、P. B. Godfrey 和 M. Schapira，“PCC：为一致的高性能重新设计拥塞控制”，载于第 12 届 USENIX 网络系统设计与实现研讨会（NSDI 15），2015 年，第 395–408 页。 [在线]。可访问：https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/dong [20] M. Alizadeh、A. Kabbani、T. Edsall、B. Prabhakar、A. Vahdat 和 M. Yasuda，“少即是多：以少量带宽换取数据中心超低延迟”，载于第 9 届 USENIX 网络系统设计与实现研讨会（NSDI 12），2012 年，第 19–33 页。 [在线]。可访问：https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/alizadeh [21] M. Alizadeh、S. Yang、M. Sharif、S. Katti、N. McKeown、B. Prabhakar 和 S. Shenker，“pFabric：最小化且近似最优的数据中心传输”，载于 ACM SIGCOMM 2013 会议论文集，2013 年，第 435–446 页。[22] C. Wilson、H. Ballani、T. Karagiannis 和 A. Rowstron，“迟到总比不到好：满足数据中心网络中的截止期限”，载于 ACM SIGCOMM 2011 会议论文集：计算机通信的应用、技术、架构与协议，2011 年，第 50–61 页。[23] B. Vamanan、J. Hasan 和 T. N. Vijaykumar，“截止期限感知的数据中心 TCP（D2TCP）”，载于 ACM SIGCOMM 2012 论文集，2012 年，第 115–126 页。[24] S. Rajasekaran、S. Narang、A. A. Zabreyko 和 M. Ghobadi，“MLTCP：一种近似集中式流调度的分布式机器学习技术”，载于第 23 届 ACM 网络热点研讨会（HotNets 24）论文集，2024 年。[25] Y. Peng、Y. Zhu、Y. Chen、Y. Bao、B. Yi、C. Lan、C. Wu 和 C. Guo，“用于加速分布式 DNN 训练的通用通信调度器”，载于第 27 届 ACM 操作系统原理研讨会（SOSP 19）论文集，2019 年，第 16–29 页。[26] T. Gale、D. Narayanan、C. Young 和 M. Zaharia，“MegaBlocks：采用混合专家的高效稀疏训练”，载于机器学习与系统论文集，第 5 卷，2023 年。[在线]。可访问：https://proceedings.mlsys.org/paper files/paper/2023/hash/5a54f79333768effe7e8927bcccffe40-Abstract-mlsys2023.html [27] G.-I. Yu、J. S. Jeong、G.-W. Kim、S. Kim 和 B.-G. Chun，“Orca：面向基于 Transformer 的生成模型的分布式服务系统”，载于第 16 届 USENIX 操作系统设计与实现研讨会（OSDI 22），2022 年，第 521–538 页。 [在线]。可访问：https://www.usenix.org/conference/osdi22/presentation/yu

[28] A. Agrawal、N. Kedia、A. Panwar、J. Mohan、N. Kwatra、B. S. Gulavani、A. Tumanov 和 R. Ramjee，《借助 Sarathi-Serve 驯服大语言模型推理中的吞吐量-延迟权衡》，载于 第*18*届 *USENIX* 操作系统设计与实现研讨会（*OSDI 24*），2024年，第 117–134 页。 [在线]。链接：https://www.usenix.org/conference/ osdi24/presentation/agrawal [29] Y. Zhong、S. Liu、J. Chen、J. Hu、Y. Zhu、X. Liu、X. Jin 和 H. Zhang，《DistServe：拆分预填充与解码，实现面向有效吞吐量优化的大语言模型服务》，载于 第*18*届 *USENIX* 操作系统设计与实现研讨会（*OSDI 24*），2024年，第 193–210 页。 [在线]。链接：https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin [30] P. Patel、E. Choukse、C. Zhang、A. Shah、I. Goiri、S. Maleki 和 R. Bianchini，《Splitwise：利用阶段拆分实现高效的生成式大语言模型推理》，载于 *2024*年 *ACM/IEEE* 第*51*届年度国际计算机体系结构研讨会（*ISCA 24*），2024年。