北京大学
PEKING UNIVERSITY

基于在网计算的分布式系统加速方法

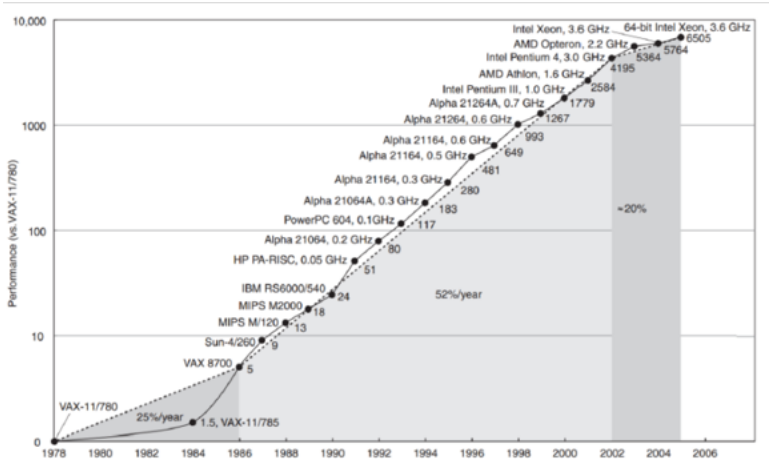
吴文斐

2023年5月

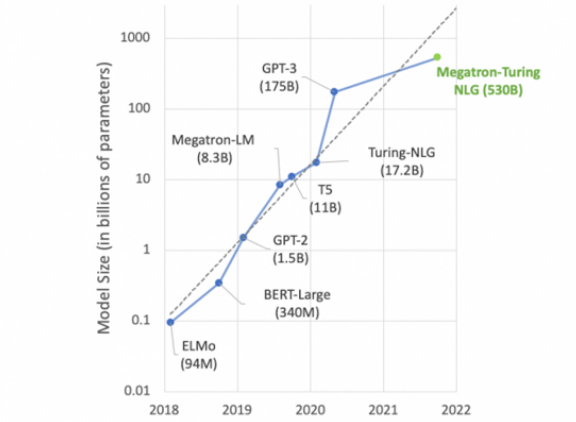
在网计算是后摩尔时代提升算力的有效方法

- 摩尔定律失效，单点计算能力达到极限；算力需求指数增长，形成矛盾
- 分布式系统扩展面临功耗墙、存储墙，搬动数据开销达到整体70%

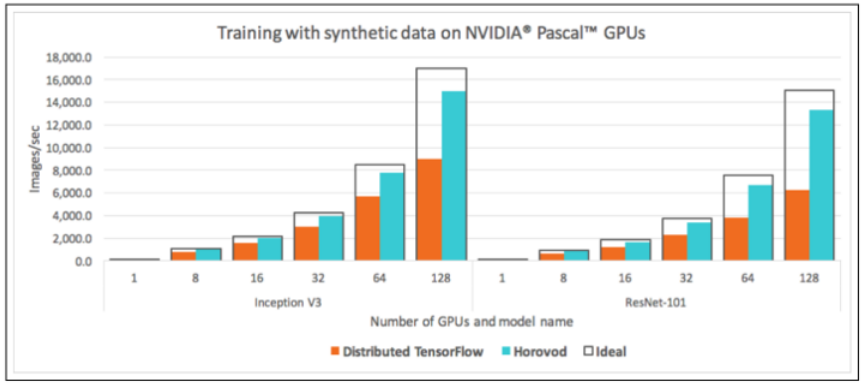
在网计算： 打破网算边界，加速传输计算过程，进而加速整体系统



摩尔定律失效



GPT-3模型有1750亿参数，
估计需要需要5000张以上的显卡



默认的TensorFlow扩展到128卡时，
显卡使用率已经降至50%

在网计算定义

在网计算是将标准应用卸载到网络设备上的计算模式。

In-Network Computing is the offloading of standard applications to run within network devices.

“应用”的概念和范畴尚需要定义。

近年来快速发展（得益于可编程网络设备的发展）

- Barefoot交换机（2013）
- Juniper Trio交换机（SIGCOMM22）
- 其他Cisco Sillon One，Broadcom Trident

在网计算的意义

提升“应用”效率（本讲座范畴）

- 优良特性
  - 压缩网络流量
  - 低时延通信
  - 高速计算
- 适用范围：分布式应用加速
  - 分布式大模型训练
  - 分布式数据分析
  - 分布式存储系统

网络功能（非本讲座内容）

可编程交换机最初为网络管理员设计，而非应用开发者。基于可编程交换机可以为“网络管理”开发一系列解决方案，但是并不属于本课题范围。

在网计算的发展空间和路线（1/2）

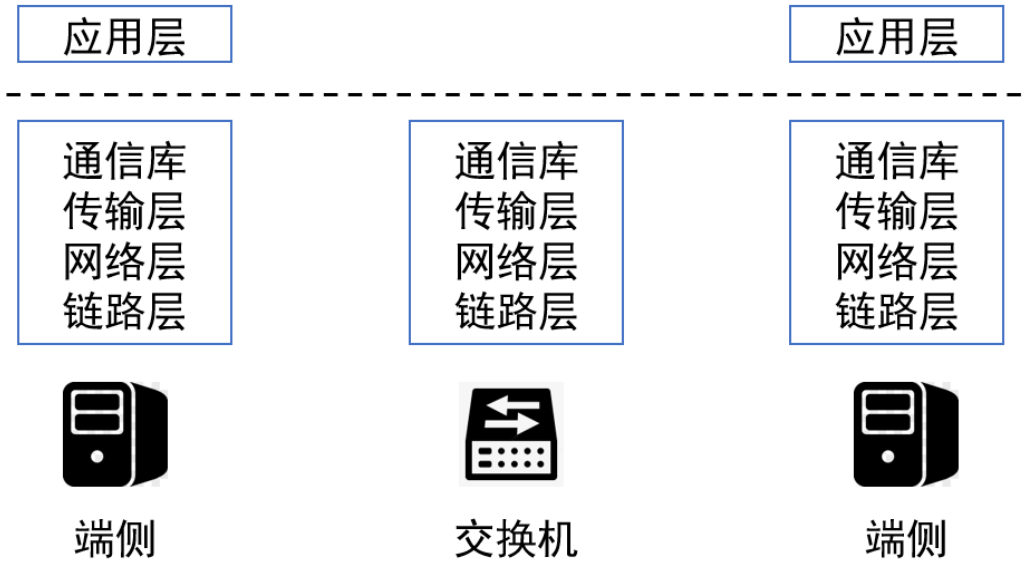
传统网络的现状

- 分层结构
  - 优势：各层独立，各自优化，形成生态（设备、运营商、操作系统、应用）
  - 劣势：性能无法达到最优
- 应用异构性较大，形成各自的通信模式
  - 高性能计算（MPI）
  - 机器学习（集合通信）
  - 大数据系统（键值计算）
  - 等等

在网计算的发展空间和路线（2/2）

在网计算发展空间大

- 大型集群计算成为新趋势（大模型、大数据）
  - 集群归一方所有，
    分层设计的前提不再具备
- 各层具备可编程性、可定制性
- 降本增效的收益更加明显
  - 集群建设成本高昂，提升效率意义重大



在网计算的发展路线

- 打通二至五层，各应用领域独立设计；在各个应用领域验证并取得收益
- 领域间逐步整合，重新设计通信库标准；使应用开发者迁移到新的开发模式
- 功能下沉，更新设备，进一步提升性能；促进大规模商业化生产

在网计算的目前发展情况

应用网络联合设计（案例）

机器学习、大数据分析、分布式存储

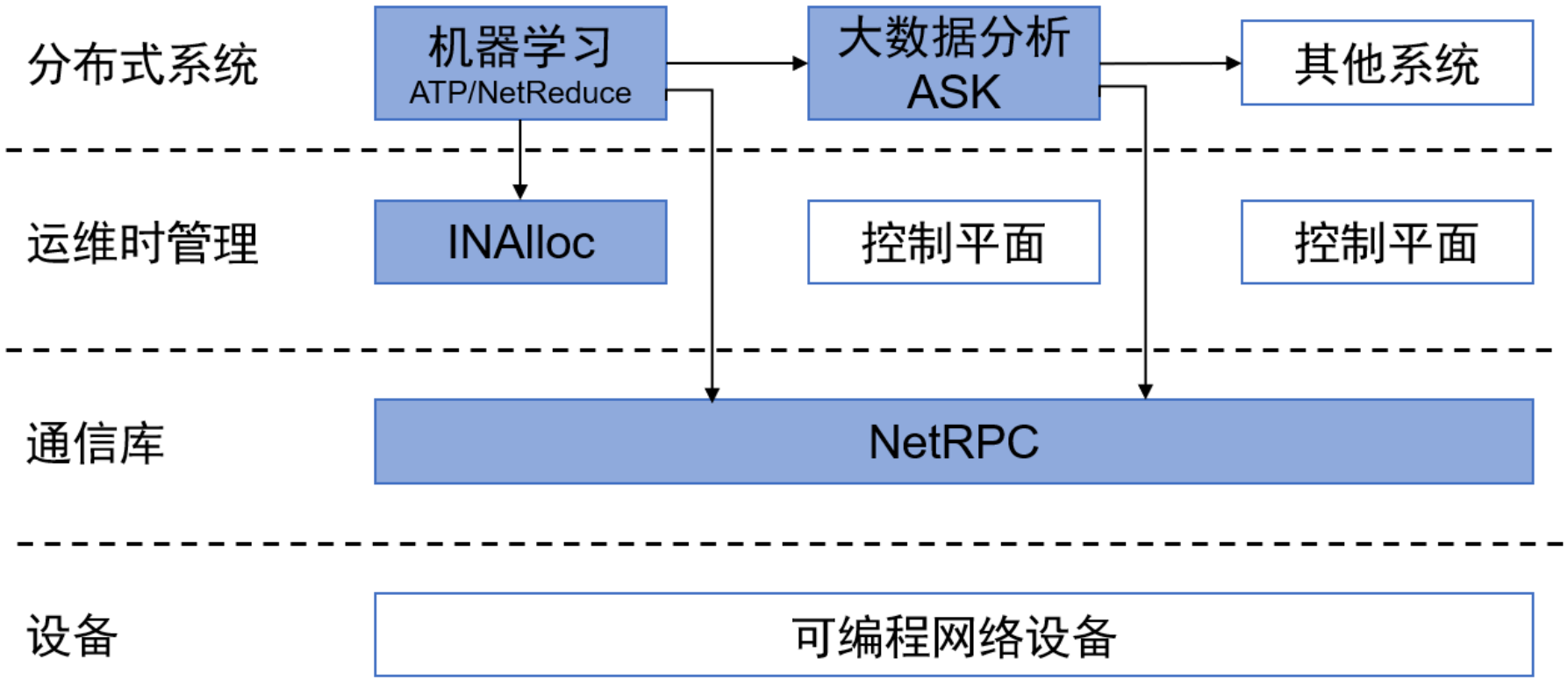
通用编程框架（开发）

多数面向网络管理员、少数面向应用开发者

运行时管理系统（部署）

交换机内存管理

路线图和我们的实践


分布式系统
机器学习
ATP/NetReduce
大数据分析
ASK
其他系统
运维时管理
INAlloc
控制平面
控制平面
通信库
NetRPC
设备
可编程网络设备

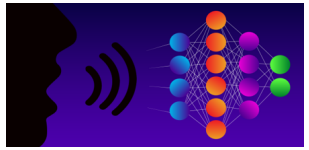
内容

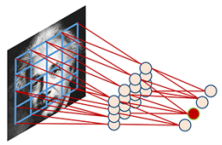
- 在网计算背景和发展路线
- 成果分享
  - 加速分布式机器学习（NetReduce，ASPLOS23；ATP，NSDI21最佳论文）
  - 分布式机器学习任务管理（INAlloc，INFOCOM23）
  - 加速大数据分析系统（ASK，ASPLOS23杰出论文）
  - 通信库设计（NetRPC，NSDI23）
- 总结

在网计算加速机器学习：背景

机器学习应用

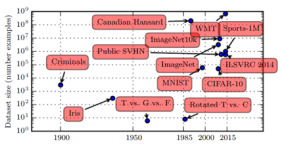
- 自然语言处理
- 计算机视觉
- 智能运维
- ……

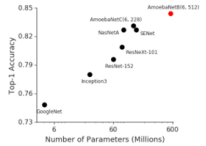


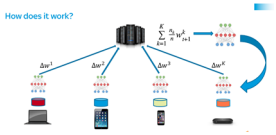


分布式机器学习

- 增长的数据集
- 增长的模型
- 场景要求（联邦学习）







机器学习训练算法

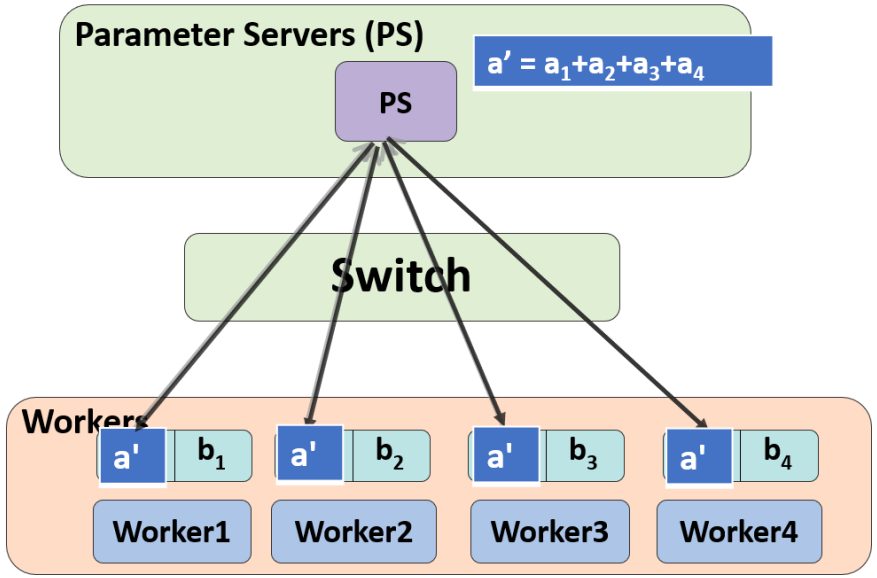
- 重复迭代至收敛
  - 计算梯度
  - 更新模型
- 分布式训练（数据并行）
  - 每个worker计算梯度
  - 所有梯度聚合并返回给worker（也
  - 更新模型

$$W^{(t+1)} := W^{(t)} - \alpha \nabla J(W^{(t)}, D^{(t)})$$

$$W^{(t+1)} := W^{(t)} - \alpha \sum_{p=1}^{P} \nabla J(W^{(t)}, D_p^{(t)})$$

分布式训练参数服务器（Parameter Server，PS）

- 假设：N个worker，梯度大小为M
- 通信量
  - worker：M
  - PS：$N \times M$



问题：PS链路成为瓶颈

- 拖慢训练速度若干倍
  - 实测VGG16会慢4倍

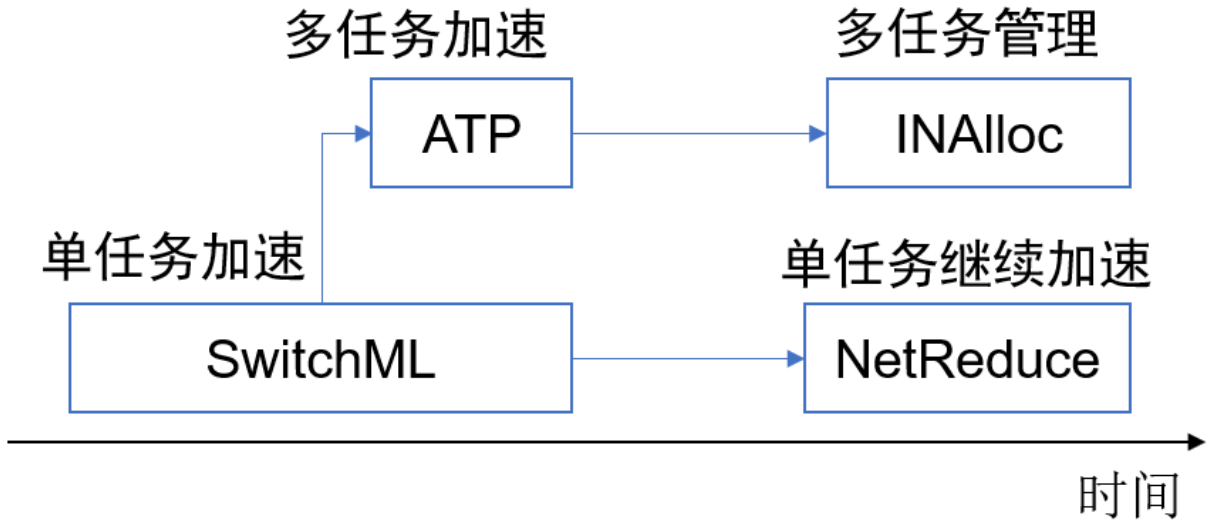
设计目标

- 主要目标：使用在网计算加速分布式训练
- 其他目标（体现在不同系统的设计中）
  - 兼容性
  - 多任务
  - 跨机柜
  - ……

	方案一	方案二
单任务加速	是	是
传输层兼容性	是	否
多任务效率	否	是
跨机柜部署	否	是

不同系统面临不同的实际场景，需求不同，目标也有差异，这导致它们设计上的差异。

研究工作间的关系


多任务加速
多任务管理
ATP
INAlloc
单任务加速
单任务继续加速
SwitchML
NetReduce
时间

方案一：交换机取代PS（NetReduce，ASPLOS23）

- 目标
  - 主要目标：使用在网计算加速分布式训练（SwitchML，NSDI21）
  - 其他目标：与RDMA兼容（NetReduce）

如果PS是服务器，如何工作？

1. 每个worker把数据发送到PS
2. PS把数据求和
3. PS将结果广播给worker

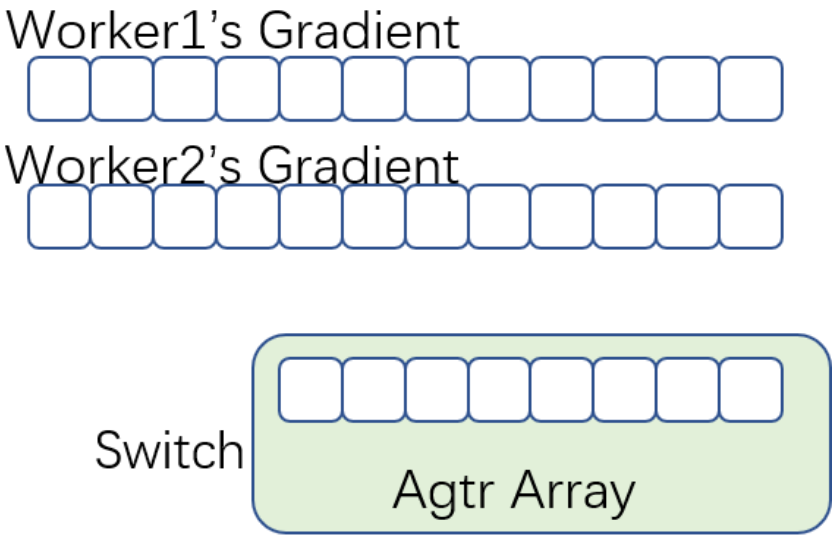
【动画】

问题：交换机能完成上述过程吗？

- 交换机能流式处理报文，但是不能缓存大块数据
- 交换机没有四层协议保证数据块完整

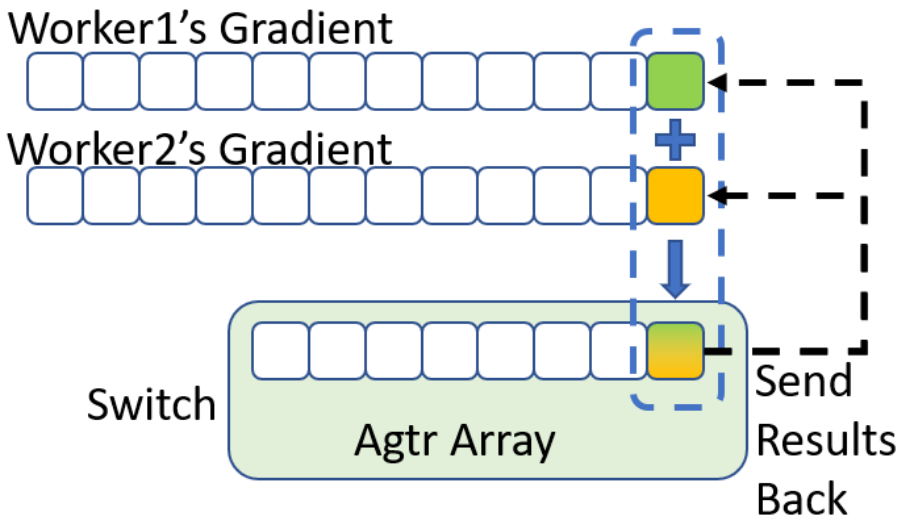
设计：体系结构和数据结构

- Worker：梯度消息被组织为一个报文序列
  - 维护一个滑动窗口发送报文，假设窗口最大值为W
- 交换机：内存组织为一个聚合器数组，大小为N



工作流程

- Worker不断发送窗口内的报文
- 报文抵达交换机后，进行聚合器寻址
  - 采用取模运算 $agtr.idx = PSN\%N$
  - 意味着 $W \leq N$ （必要不充分）
- 一个聚合器中的聚合完成（通过bitmap判断），将结果组播给workers
  - 携带聚合结果的报文被认作ACK
  - 释放$(i+W)\%N$位置的聚合器（后文解释）
- 结果报文ACK使worker的滑动窗口继续滑动，发送新的报文



【动画】

问题：如果网络丢包怎么办？

计算要求正确可靠，但是网络不稳定

- Worker端，如果没有收到ACK（或者重复ACK），重传数据报文
- Switch端，可能收到重复的报文，但是不应该重复计算（正确性的要求）
  - 聚合器的bitmap记录每个worker的报文是否已经参与计算
  - 收到重传的梯度报文，不会重复计算

难点：交换机内存的循环使用

- 聚合器数组可能小于消息大小，如何循环使用聚合器数组？
  - 即聚合完成后，该何时释放聚合器？
- 如果在结果组播给workers时释放聚合器，是错误的。
  - 无错worker：未丢失ACK，窗口前移，不会重传，发送新报文；
  - 出错worker：丢失ACK，窗口不动，继续重传，发送旧报文；
  - 交换机中，新旧报文均无法完成聚合，协议存在死锁。

【动画】

难点：交换机内存的循环使用

- 解决方法：一个报文需为“一个窗口”外的报文初始化聚合器
  - 第i个报文去释放未来第$i+W$个报文的聚合器
- 意味着 $N \geq 2W$ （充分且必要）
  - 第i个报文出现时，$(i-W, i+W)$内的报文均可能存在在聚合器数组中，
  - 需要被释放的$(i+W)\%N$必须不与上述区间重叠
  - 共需要至少$2W$个聚合器。

符号	释义
N	聚合器数组大小
W	发送窗口最大值

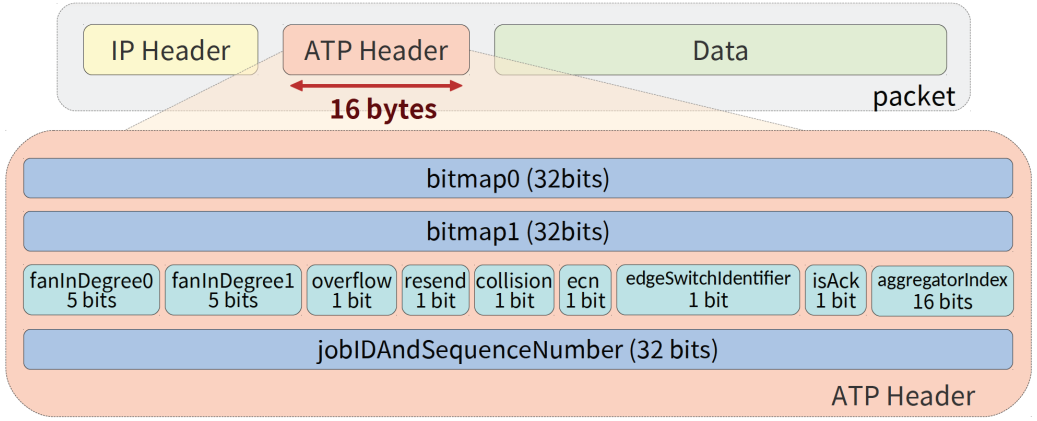
注：SwitchML (NSDI21) 中的Shadow Copy技术与此思路类似，也需要窗口两倍的交换机内存。

以上是无PS的在网计算主要工作流程

- 在SwitchML、NetReduce、ATP等均有设计

端侧协议栈需要的功能

- 传输层功能
- 指导交换机聚合
- 北向接口 `AllReduce()` 、 `PushPull()` 等是单边的，不同于TCP的双边 `send()` 、 `recv()`



次要目标：传输层透明（NetReduce，ASPLOS23）

- 问题描述：目前的方案替换了终端TCP/IP协议栈，造成部署中的问题
  - 性能：不能受益于最新的传输方法
    （RDMA over Converged Ethernet，RoCE）
  - 开发：重复实现传输层的功能
  - 管理：部署需要隔离、网络流量需要隔离

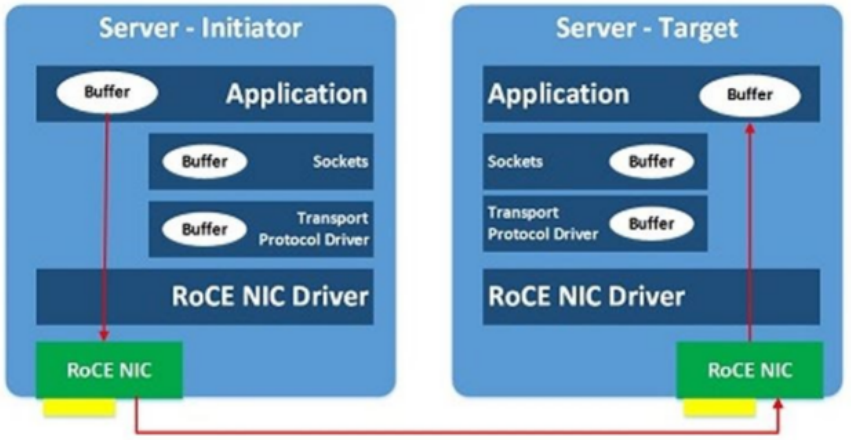


Table 1: Performance of RoCE, parallel DMA, and DPDK.

	CPU	Throughput (Gbps)	RTT (us)
RoCE	25.6%	84.2	5.7
Parallel DMA	100%	58.67	70.2
DPDK	200%	90.5	20.5

Table 2: Lines of code of functions in ATP network stack.

Function	Lines of Code	Percentage
Packetization & IO	1090	32.12%
Flow Control	50	1.47%
Reliability	181	5.33%
Congestion Control	64	1.89%
Floating Point Support	220	6.48%
Fallback	100	2.95%
Others	1689	49.76%
Total	3394	100%

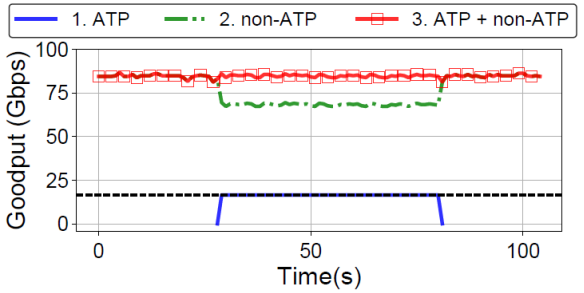
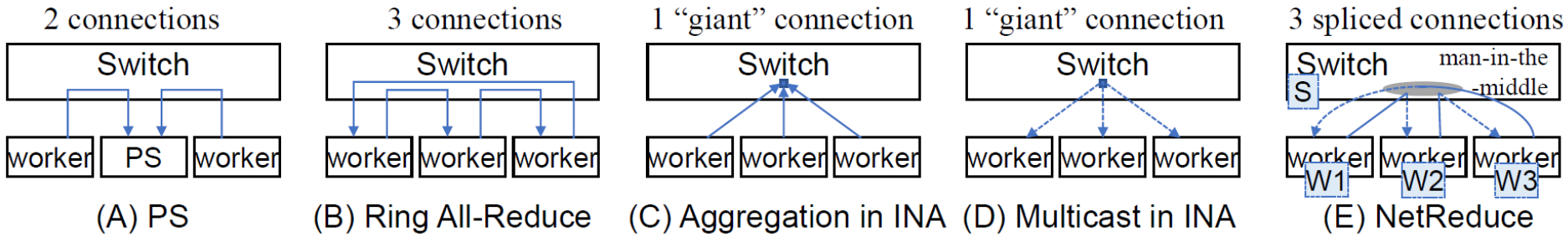


Figure 2: Bandwidth contention of VGG16 in ATP and background DCTCP, cited from [35].

问题：能否实现传输层透明的在网聚合？

- 思路：交换机内部实现网络流拼接，使终端无法察觉报文改动



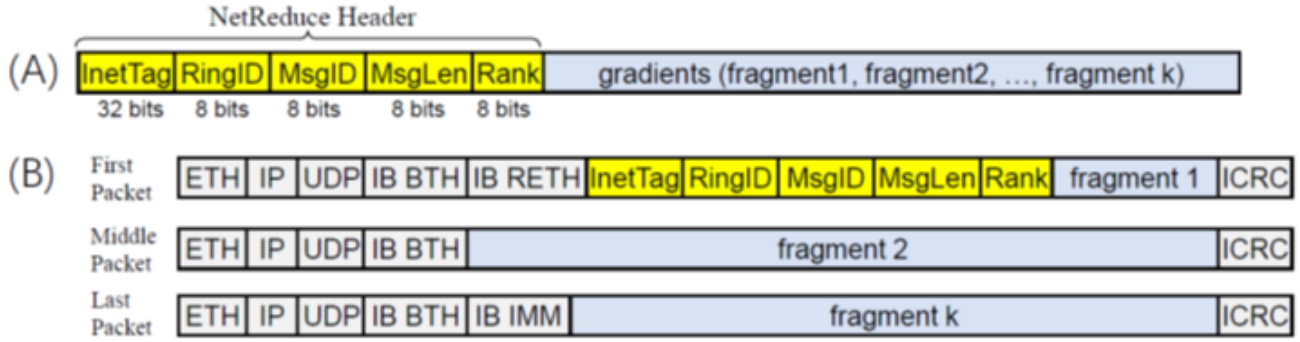
体系结构和工作流程

- 端侧：保持RoCE协议栈不变
  - 添加流量控制模块，发送梯度消息
  - 按照环状建立逐跳RDMA连接
- 交换机外挂FPGA加速器
  - 聚合器数组大小为N
    - 聚合器内缓存报文

【动画】

- 工作流程
  - 发送端：使用流控模块发送报文
  - 交换机：用模运算进行寻址
    - 多个连接的报文聚合为一个
    - 恢复报文头，发送回原连接
  - 接收端：接收报文

难点：INA报文头恢复



报文格式

- 消息格式（右图A）：从主机到网卡
- 报文格式（右图B）：网卡切包

问题：交换机需要INA报文头，但是RoCE不提供每报文添加报文头的灵活度

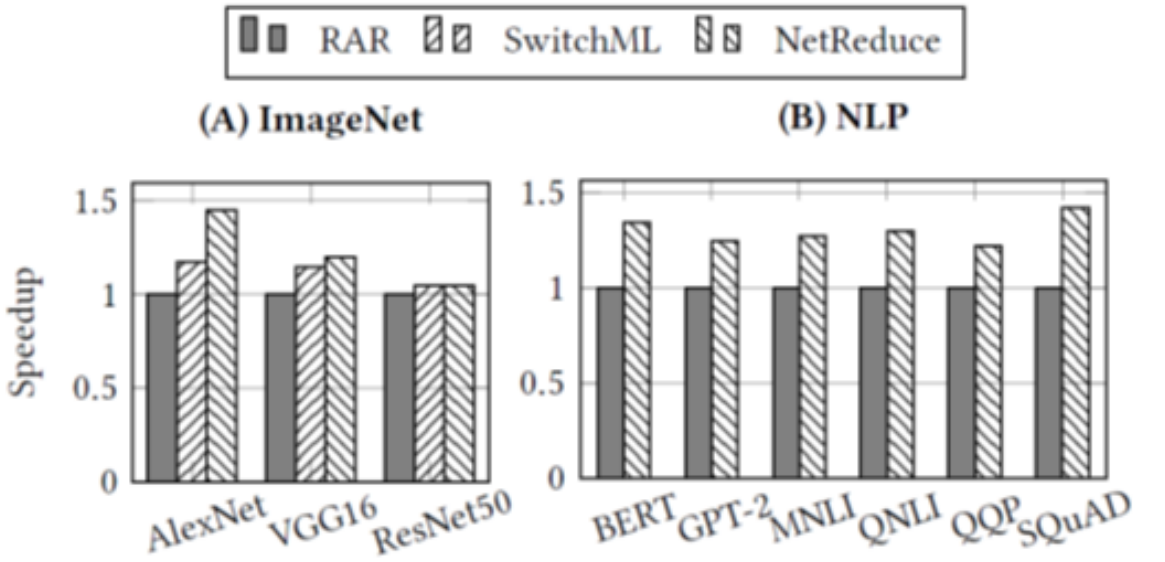
解决方法：连接查询表 Connection Lookup Table

- 维护连接和INA状态之间的映射关系
  - <SrcIP, DstIP, DstQP>:<RingID, MsgID, MsgLen, Rank>
- 每个消息的第一个报文填表
- 每个消息的其他报文查表，获得INA状态

【动画】

实验结果（NetReduce）

- NetReduce提升训练效率，获得在网聚合和RDMA双重性能增益
  - 训练AlexNet比Ring AllReduce快45%



方案一小结

- 可以用交换机替换PS
- 可以做到对传输层透明（意味着可以与RDMA集成）
- 交换机内存为端侧窗口两倍
- 其他特性（详情后续讨论，或者参考论文）
  - 交换机不支持浮点数计算，需要端侧做浮点和整数之间的转换（SwitchML）
  - 扩展到分层聚合：TOR交换机不变，高层交换机为树状聚合（NetReduce）
  - 扩展到并发聚合：多机多卡时，可以形成多个并发聚合任务（NetReduce）

注：NetReduce可以在Tofino上实现，把RDMA连接信息写到Match-Action表中即可

方案二：新背景

生产环境中的训练任务

- 多租户
  - 多任务共享基础设施
- 多机柜
  - 例如，BERT-Large训练跨多机柜

Time	System	Number of Nodes	Number of V100 GPUs
47min	DGX SuperPOD	92 DGX-2H	1472
67min	DGX SuperPOD	64 DGX-2H	1024

目标

- 主要目标：使用在网计算加速分布式训练
- 次要目标：
  - 支持多租户任务
  - 支持跨机柜部署

解决方案：聚合传输协议设计ATP

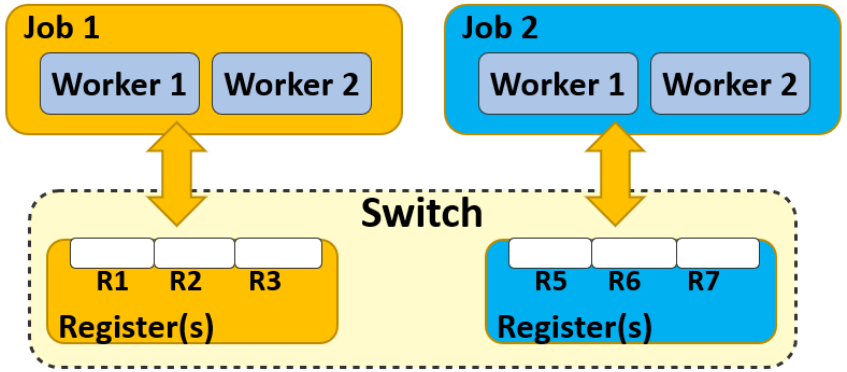
Aggregation Transmission Protocol，ATP

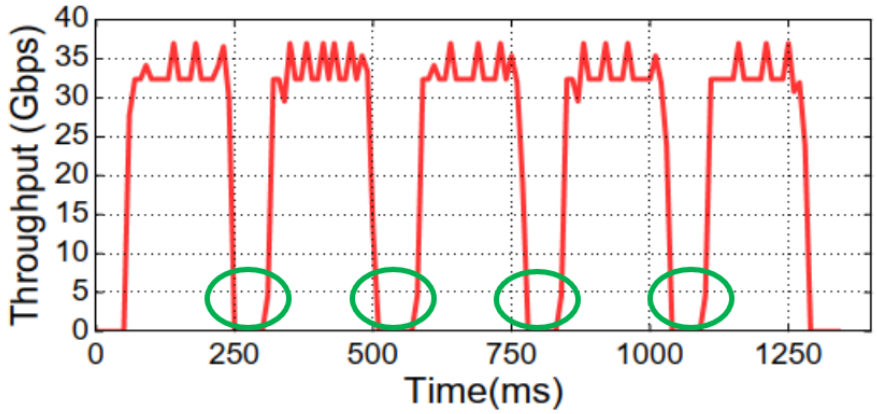
特性如下：

- 在网计算加速分布式训练
  - 可靠性
- 多租户支持
- 拥塞控制
- 支持跨机柜
- 浮点数支持

多租户多任务：静态交换机内存分配

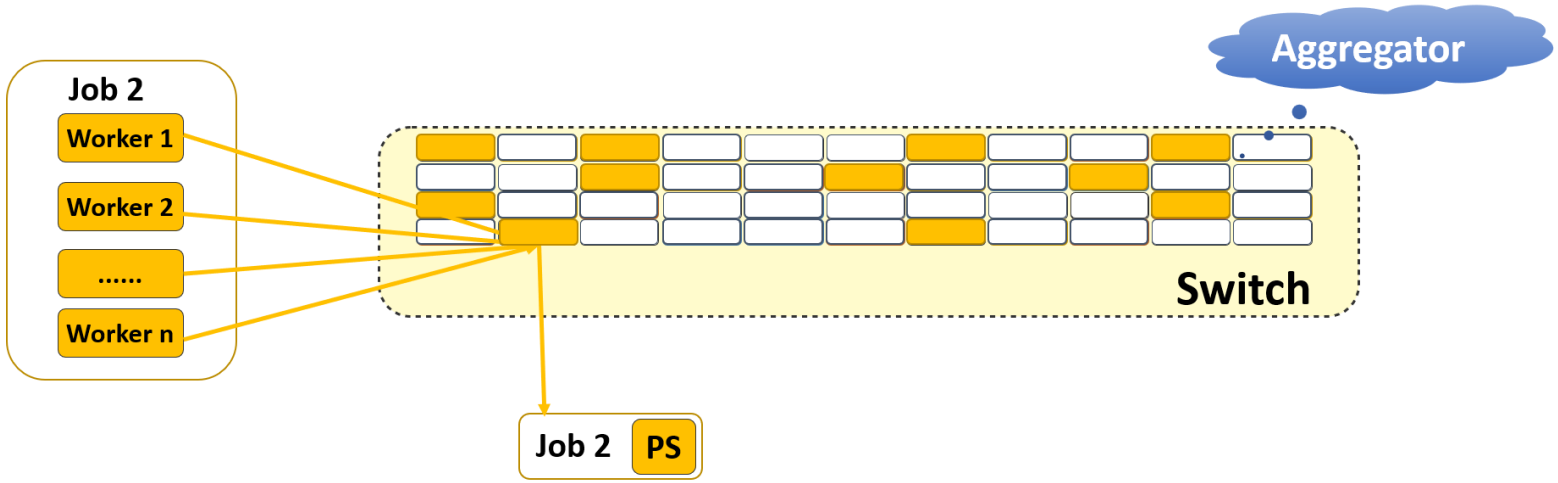
- 划分交换机内存为隔离的区域，分配给多任务
- 不足之处
  - 交换机内存低效率：机器学习周期性计算和传输，交换机资源闲置
  - 集成复杂度：需要将交换机管理与任务管理器集成





动态交换机内存分配

- 交换机内存为一个聚合器资源池
  - 对于每个任务的报文采用先到先服务（FCFS）
- 去中心化聚合器寻址：$Agtr.idx = Hash(JobID, PSN)$



体系结构（较方案一改动部分）

- 保留PS，因为
  - 寻址可能失败，需要回退机制（Fallback）
  - 交换机难以正确处理重传报文
- 聚合器
  - 记录JobID、Seq，检测冲突
  - 释放需要由ACK完成

工作流程（理想情况）

- worker端维护滑动窗口，发送梯度报文
- 梯度报文抵达交换机，进行寻址
  - 寻址成功，聚合器计算；聚合完成时，结果发送至PS
  - 寻址失败，透传至PS
- PS接到聚合结果或者透传报文，完成聚合，返回ACK（携带结果）
- ACK抵达交换机，释放聚合器
- ACK被组播给workers，推动滑动窗口

【动画】

方案一中报文seq释放“一个窗口”外的聚合器hash(JobID, seq+W)的做法可能出错，因为别的任务可能在使用该聚合器。

问题：网络丢包时如何处理？

worker：没有收到ACK时重传数据包

PS：收到完整结果后返回ACK

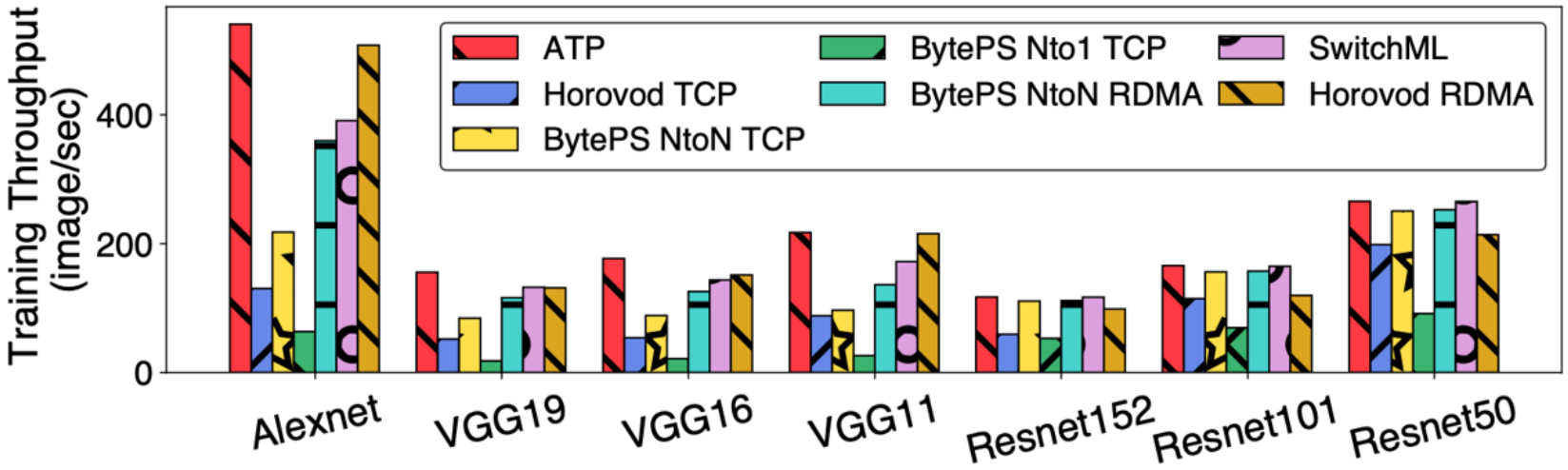
交换机？

- 重传的包不能开启聚合，避免卡死
- 重传的包将结果带到PS，尽快清空聚合器

【动画】

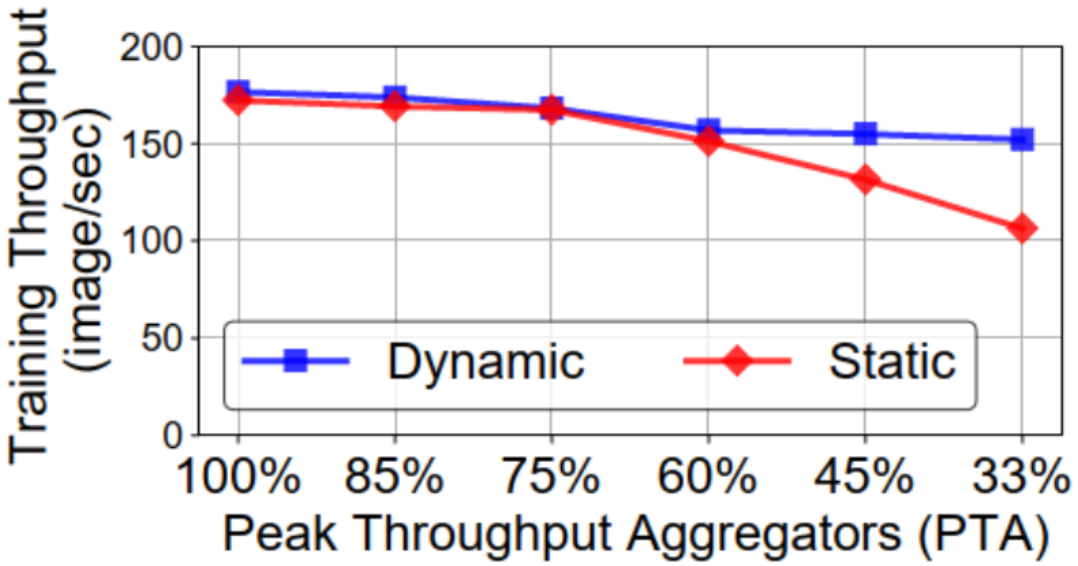
单任务性能

- 指标：平均每台服务器每秒训练图片数量image/second
- ATP解决了传输瓶颈
- ATP对于通信瓶颈任务增益更大
- ATP稍好于Horovod，仅消耗一半网络带宽



多任务性能：动态内存分配v.s.静态内存分配

- 三个VGG16任务，静态分配中三个任务平分内存
- 调整交换机内存使所有任务带到吞吐量峰值
  - 称为吞吐峰值聚合器（Peak Throughput Aggregator，PTA）
  - 降低PTA使三个任务产生竞争
- 动态内存分配下的吞吐下降更慢



方案二小结

- 统计复用交换机，利用率更高
- 正确性机制更复杂
- 其他特性（详情后续讨论，或者参考论文）
  - 分层聚合：叶节点交换机不变，高层需要处理要避免部分聚合
  - 浮点数计算：能额外处理溢出的情况，交给PS处理
  - 拥塞控制：ECN聚合+AIMD机制

方案一和方案二的对比

	方案一	方案二
组成	交换机	交换机+PS
寻址机制	冲突避免（模运算）	冲突探测（哈希）
内存使用	2倍窗口	1倍窗口
内存分配	隔离	共享
适用范围	单任务隔离	多任务共享
优势	可预测性能，不需PS	高资源利用率

小结

- 交换机通常能显著加速系统效率
  - ATP、SwitchML、NetReduce等系统
- 网络异常情况下的正确性保证通常是协议设计的难点
  - 例如，丢包重传时计算的正确性保证
- 实际场景中的不同需求，会导致数据平面设计的不同
  - 兼容性
  - 多租户
  - 跨机柜
  - ……

目录

- 在网计算背景和发展路线
- 成果分享
  - 加速分布式机器学习（NetReduce，ASPLOS23；ATP，NSDI21最佳论文）
  - 分布式机器学习任务管理（INAlloc，INFOCOM23）
  - 加速大数据分析系统（ASK，ASPLOS23杰出论文）
  - 通信库设计（NetRPC，NSDI23）
- 总结

部署中的问题

如果有多个INC应用（实例），该如何管理网络？

历史：分时操作系统完成对多程序的资源隔离和分配

- 例如，OS/360、Multics
- 虚拟化
  - 进程：虚拟化CPU
  - 内存管理：虚拟内存

Time-sharing was first proposed in the mid- to late-1950s and first implemented in the early 1960s. The concept was born out of the realization that a single expensive computer could be efficiently utilized if a multitasking, multiprogramming operating system allowed multiple users simultaneous interactive access. [Wikipedia]

资源管理是一个研究成果很多的领域

- 应用和系统目标
  - 高性能
  - 资源利用率
  - 公平性
- 策略
  - 循环（round robin）
  - 多级反馈队列（multi-level feedback queue）
  - 等等

我们的目标

在网计算的控制平面，管理多应用实例

- 如何设计管理策略？
  - 需要与应用场景相结合
    - 策略的目标
    - 管理域的范围
- 如何使交换机资源可管理？
  - 策略实施机制（enforcement）

场景（之一）

- 生产环境中的多机器学习任务
  - 有截止时间要求（Deadline-Aware，DA Jobs）
  - 无截止时间要求（Best-Effort，BE Jobs）
- INC可以提升机器学习任务效率，但是需要消耗新的资源——交换机内存
  - 不足的交换机资源会导致训练效率下降
- 场景假设：
  - 应用和交换机控制器可以联合开发

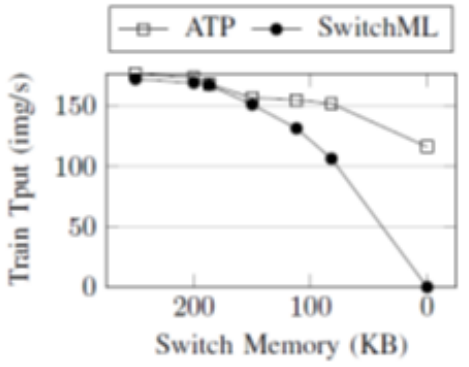


Fig. 1. Average training throughput under limited switch memory of 3 concurrent VGG16 jobs.

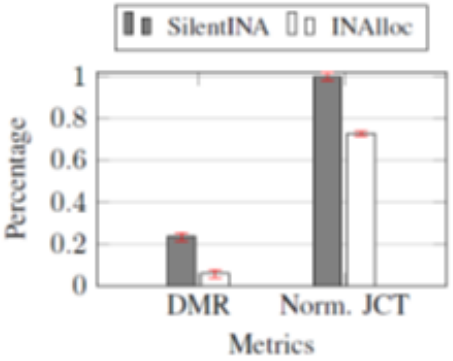


Fig. 2. [Testbed] Impact of memory management to performance, with real-world hybrid workload.

调度策略是什么？

最早截止时间优先（Earliest Deadline First）

周期性的

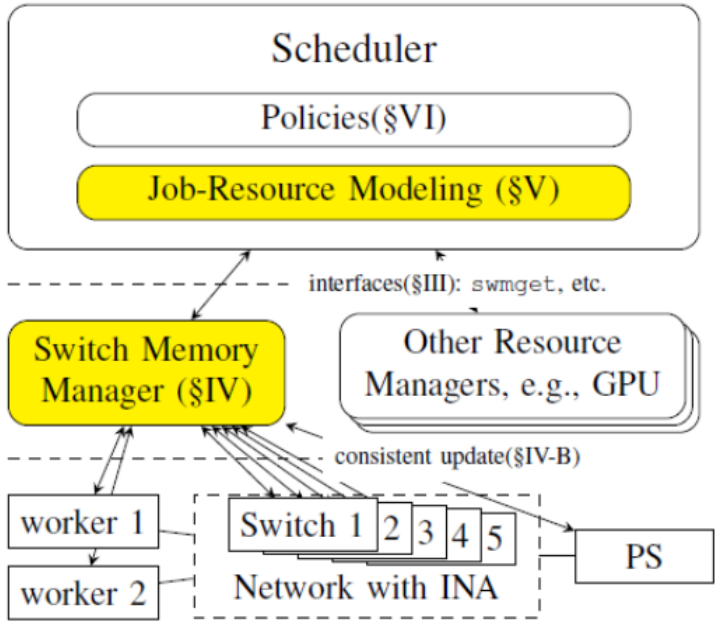
- 将任务按照截止时间排序
  - 优先将交换机内存分配给截止时间早的任务
  - 剩余内存共享给BE任务和暂时无法满足的MD任务

$$Time_{job} = Epochs \times \left(Time_{comp} + \frac{Size_{model}}{Throughput} \right)$$

$$Memory_{switch} = Throughput \times RTT$$

交换机内存管理（1/2）

- 体系结构如左图，引入交换机内存管理模块，任务时间-资源模型模块
- 交换机内存管理器
  - 北向接口：内存分配
  - 南向接口：向终端发送内存区域(offset, size)、向交换机发送规则



交换机内存管理（2/2）

- 寻址模式：因为内存可能共享，故采用哈希函数+回退机制寻址
  - $agtr.idx \leftarrow \mathbf{Hash}(PSN, JobID)\%Size + Offset$
  - 端侧计算地址，并封装在报文头部

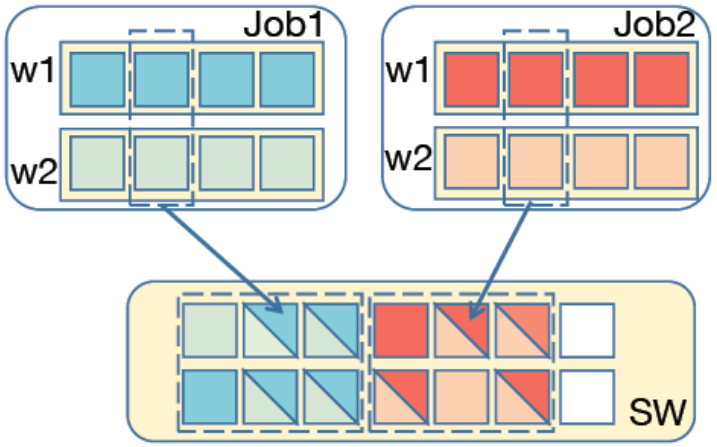


Fig. 4. Isolated memory linkage

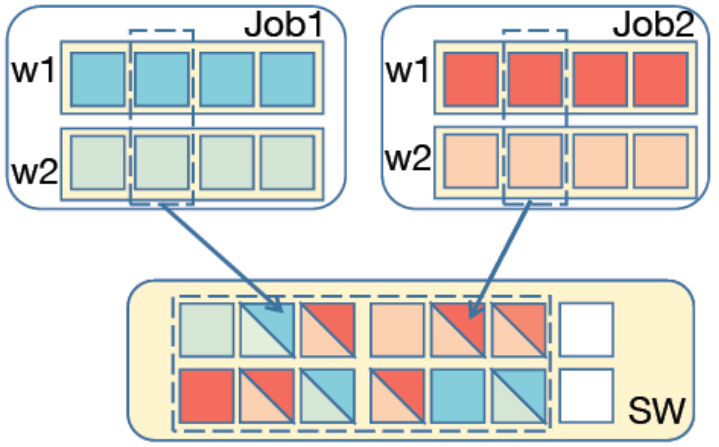


Fig. 5. Shared memory linkage

难点：一致性更新

多端侧需要一致性更新

- 一个任务有多个端侧（多个workers、一个PS）
- 周期性改变交换机内存时，所有的端侧应该在同一序列号处更新至新的内存区域

一致性更新协议

1. 控制器下发通知规则至交换机
2. 数据包携带通知到端侧
    i. 数据包序列号为PSN
    ii. 端侧选择$PSN + W$为未来更新的时刻

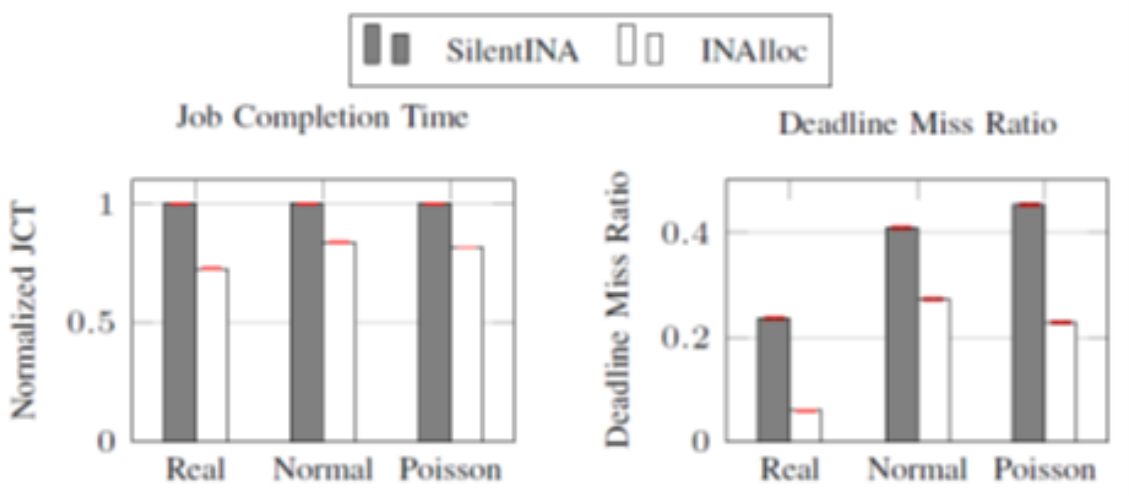
原理

所有worker都会选择$minPSN + W$，$minPSN$为规则下发后交换机遇到的最小PSN

- 每个worker的minPSN相同，因为ACK还没返回
- 每个worker计算$minPS + W$时，该包还未发出，因为在一个窗口外

实验：任务完成时间

交换机内存管理可以有效降低平均任务完成时间，提高服务满足率



小结

涉及多任务时

- 需要合理分配资源
- 需要引入资源管理模块，部署分配方案
- 改进数据平面，保证管理中的正确性

目录

- 在网计算背景和发展路线
- 成果分享
  - 加速分布式机器学习（NetReduce，ASPLOS23；ATP，NSDI21最佳论文）
  - 分布式机器学习任务管理（INAlloc，INFOCOM23）
  - 加速大数据分析系统（ASK，ASPLOS23杰出论文）
  - 通信库设计（NetRPC，NSDI23）
- 总结

大数据中存在数据流聚合

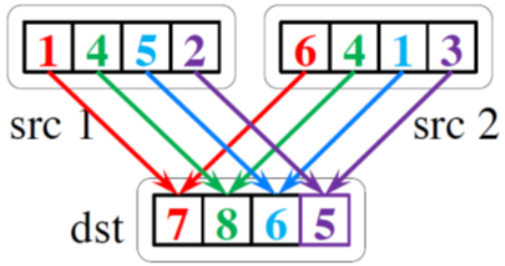
- 大数据系统中的`ReduceByKey()`
- 数据库系统中的`sum`、`count`等

大数据系统中的聚合为异步聚合

值流（value stream）

同步聚合（机器学习）

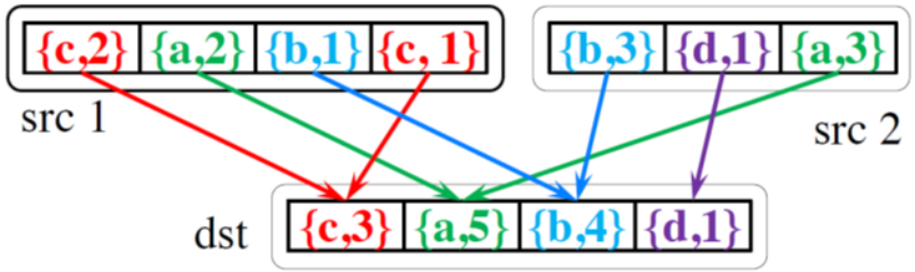
- 键（或者索引index）相同
- 键线性排列，可预知
- 键次数有界



键值流（key-value stream）

异步聚合（大数据），包含同步聚合

- 不同流中键不同
- 键无序，不可预知
- 键次数没有界



目标

基于可编程交换机，设计一套适用于键值流聚合的系统

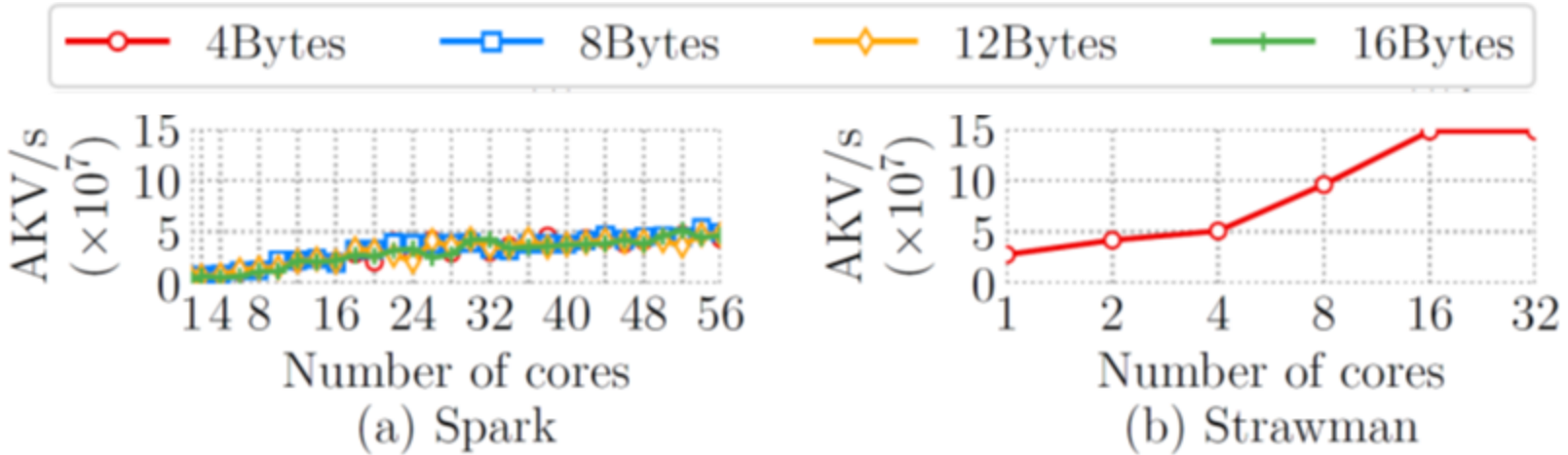
- 适配不同的应用
- 提升系统效率
- 有正确性保证

稻草人方案（Strawman Solution）

设计一个初级方案展示可行性，有三条假设：【动画】

- 每报文携带一个键值元组（交换机内存只支持单次读写）
- 网络中没有报文丢失（同步聚合中的可靠性设计不能正确工作）
- 交换机内存能够容纳所有键值元组，且预先分配key到聚合器中的地址

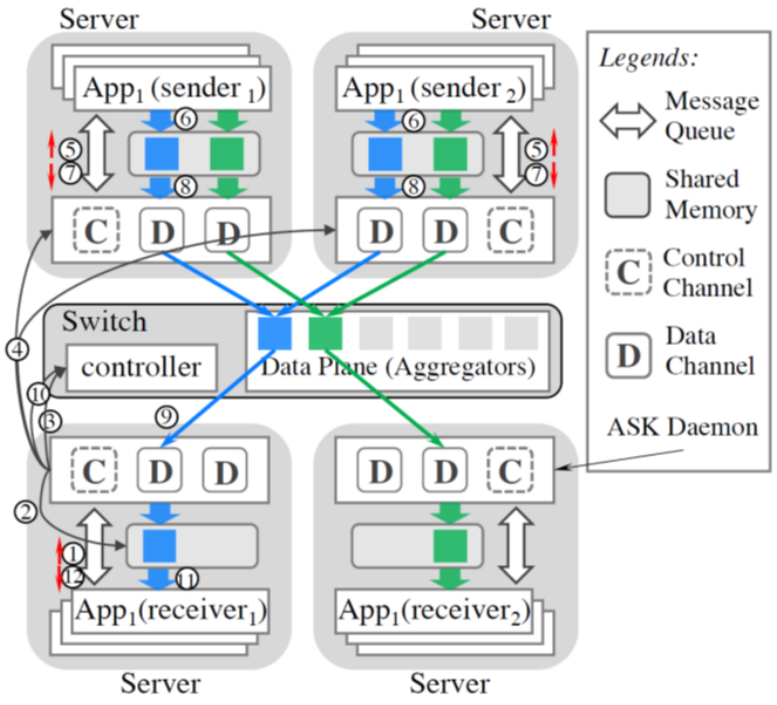
（56核单服务器）效果：提升键值聚合效率至少3倍

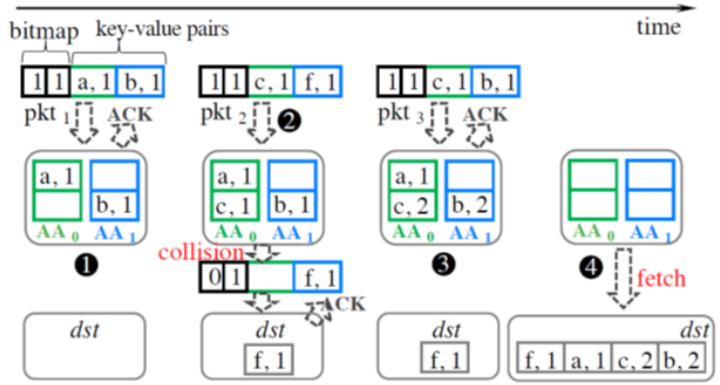


ASK体系结构和工作流程

- 端侧启动独立的agent，为应用提供服务
- 应用向服务提交请求
- agent按照FIFO处理请求
- 每个请求为一个聚合任务
  - 多个发送者、一个接收者
  - 每个包中的键值元组动态寻址

    $agtr.idx = \mathbf{hash}(key)$
  - 交换机尽力而为+端侧回退（类似ATP）
  - 任务结束后，接收端取回

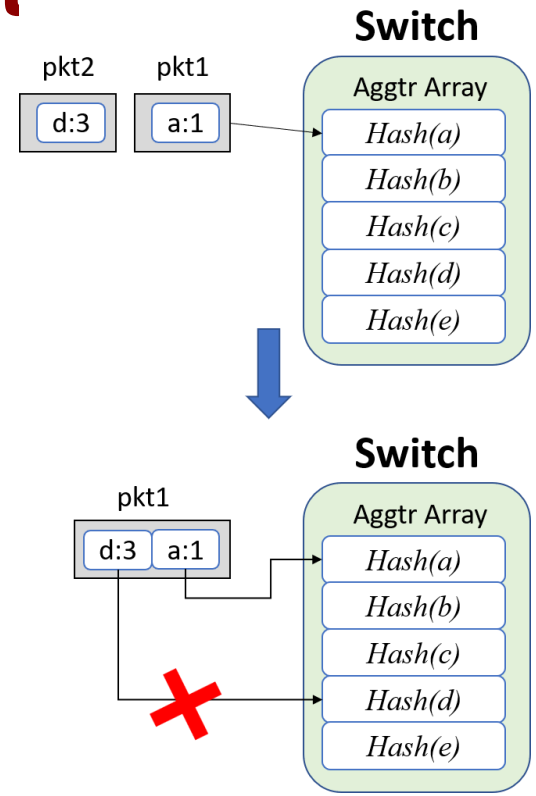




难点1：提升系统有效吞吐率（Goodput）

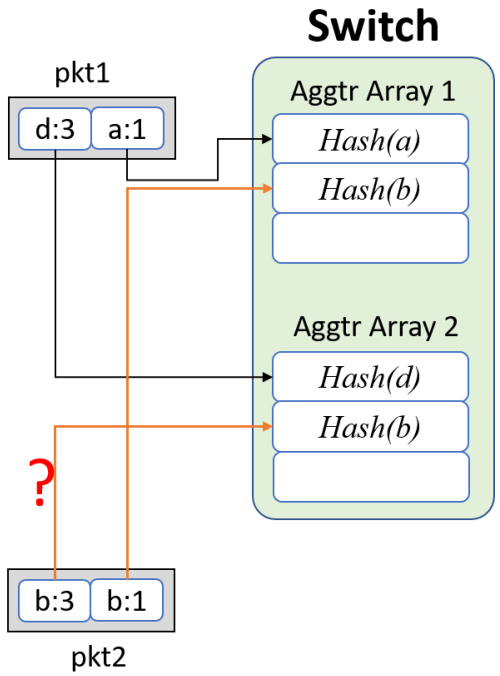
问题：有效吞吐率低下

- 有效吞吐率：100Gbps链路上为9.76Gbps
- 改进1：一个报文携带多个元组
- 新问题：交换机内存仅支持单次读写



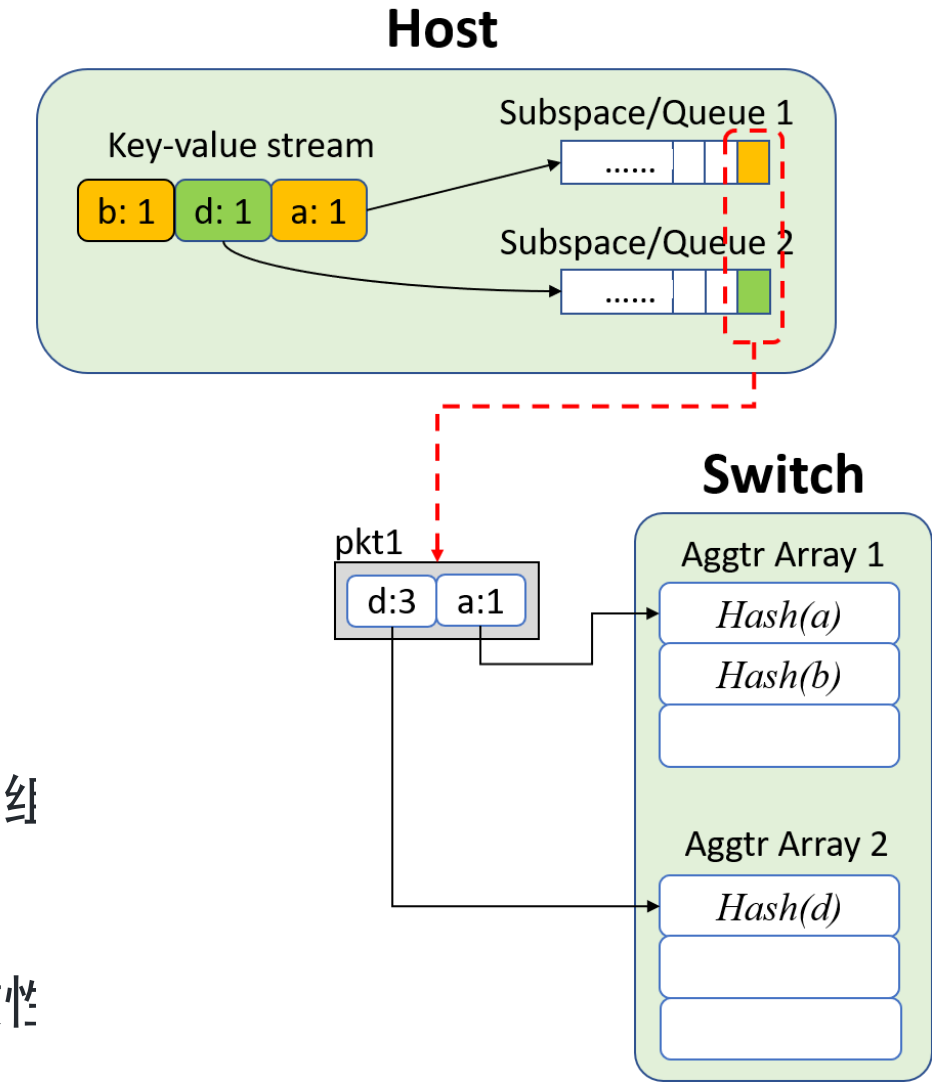
难点1：提升系统有效吞吐率

- 设计
  - 多元组报文
- 改进2：二维聚合器数组（Aggregator Array，AA）
  - 第一维度的每个AA对应报文中的一个元组
- 新问题：一个键会出现在多个AA中，占据多个位置



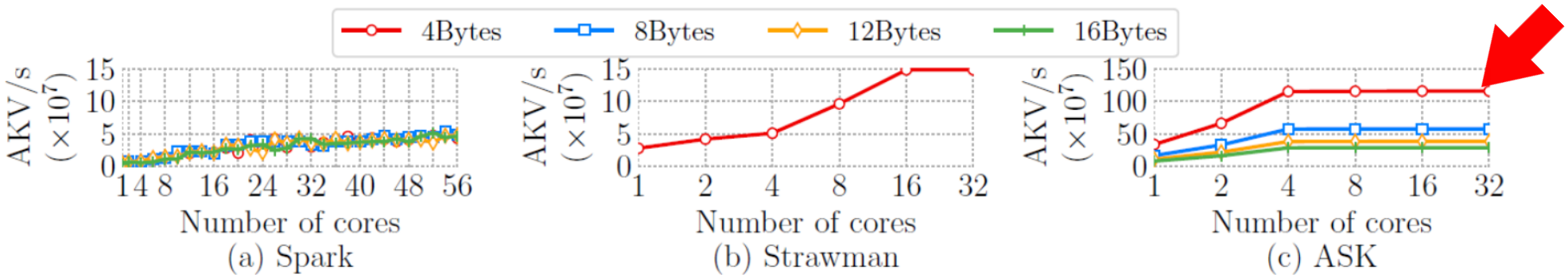
难点1：提升系统有效吞吐率

- 设计
  - 多元组报文
  - 二维聚合器数组
- 改进3：端侧元组顺序重构
  - 将键的空间划分为互不重叠的子空间
  - Hash(Flow ID) → 子空间
  - 报文有多个槽位（slot）
    - 每个槽位从每个子空间加载一个元组
    - 每个槽位被交换机中一个AA处理
    - 加一个bitmap表示报文中槽位有效性



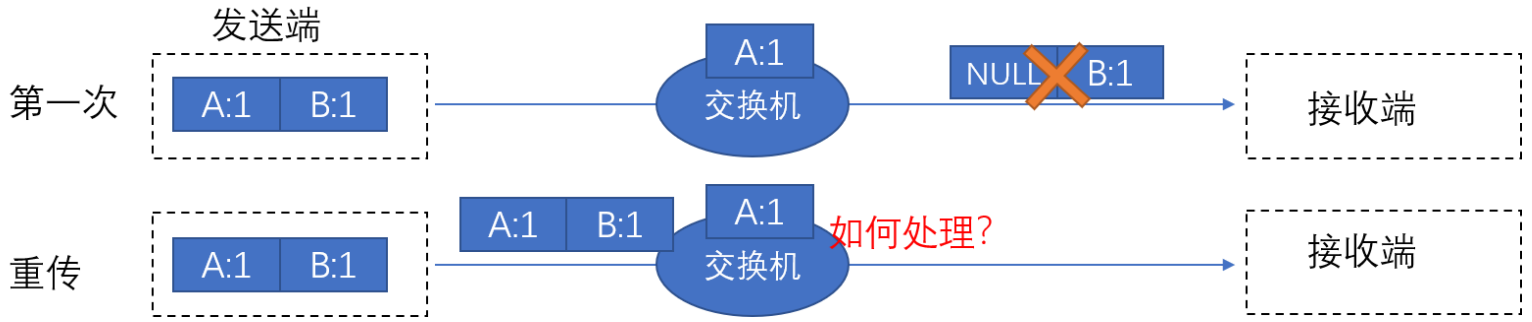
难点1：提升系统有效吞吐率

- 设计
  - 多元组报文
  - 二维聚合器数组
  - 发送端协助报文构造和寻址
- 效率：提升155倍处理速度



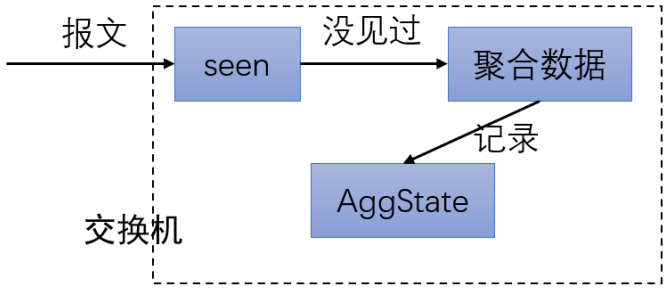
难点2：可靠性和正确性

- 一种特殊情况
  - 多元组报文可能导致一个报文被“部分聚合”（partial aggregation）
  - 部分聚合的报文抵达接收端前丢失，该报文重传至交换机时，该如何处理？
    - 丢弃？
    - 聚合？
    - 透传？
    - 以上均不对

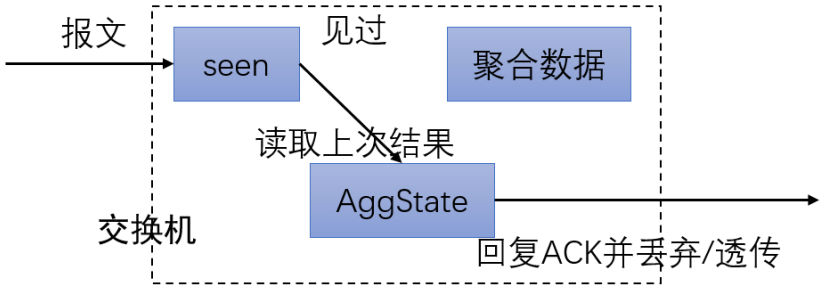


难点2：可靠性和正确性的解决方案

- 记录每一个报文在交换机中的出现与否（seen）和处理结果（AggState）
  - 一个报文的第一次出现，记录其出现，处理元组，并记录处理结果（bitmap）
  - 一个报文的后续出现，不处理元组，直接复制第一次处理结果
- 问题：状态爆炸？
  - seen和AggState采用循环数组，大小为$2W$
  - 端侧守护进程复用长连接传输多个任务，控制连接数量



报文第一次出现时正常处理并记录



报文后续出现时，复用之前的结果

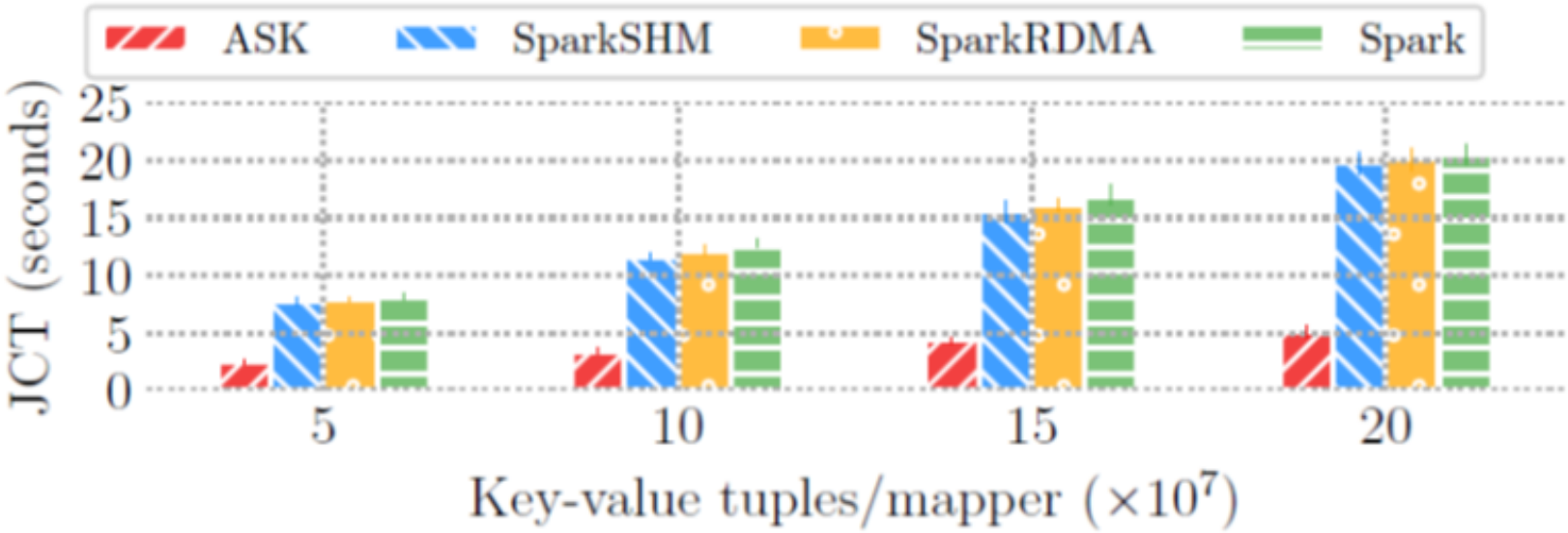
难点3：提升内存使用效率

- 现象：实际工作负载中，存在键偏斜（高频键+低频键，或者热键+冷键）；
  聚合器使用FCFS，冷键可能占据聚合器导致热键无法使用
- 问题：如何让交换机处理热键，提升效率？
- 约束：系统没有键分布的先验知识，
  不能为热键预留聚合器
- 方案：每个AA实现为双副本
  - 一旦发现某个子空间中的键大量出现在接收端
    （意味着交换机内存紧张）
    - 切换该子空间的AA至另一副本
    - 从切换前的副本中取回结果

原理：每次切换副本后，当前副本为空；所有键获得一次新的机会抢占聚合器，热键概率更高；多轮统计下来，热键在交换机中聚合次数增加

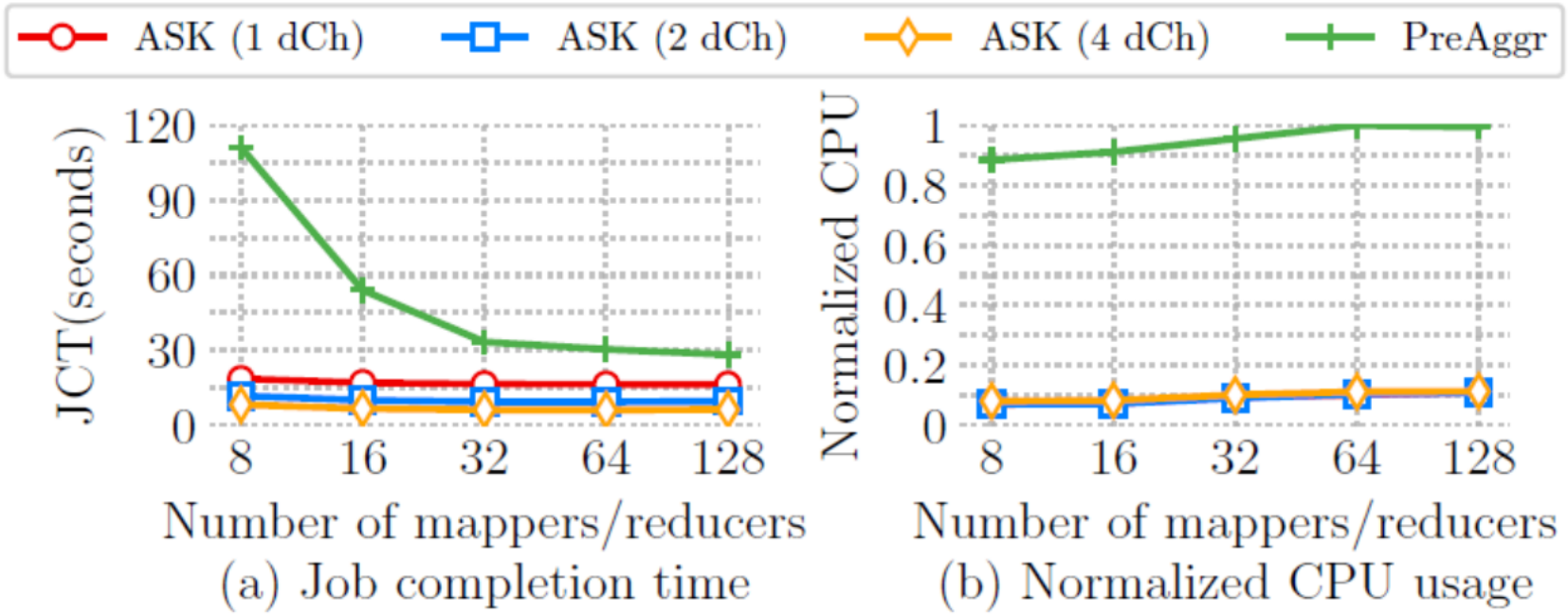
实验：任务加速

- 工作负载：词频统计（WordCount）
- 结果：降低任务完成时间



实验：CPU使用率

- ASK四核的任务完成时间比原始Spark56核任务完成时间的50%还少很多



小结

- 在网计算可以有效提升大数据分析系统效率，性能增益主要来自计算卸载
- 系统设计受硬件计算能力（不仅是性能）约束
- 正确性保证仍然是最挑战部分

目录

- 在网计算背景和发展路线
- 成果分享
  - 加速分布式机器学习（NetReduce，ASPLOS23；ATP，NSDI21最佳论文）
  - 分布式机器学习任务管理（INAlloc，INFOCOM23）
  - 加速大数据分析系统（ASK，ASPLOS23杰出论文）
  - 通信库设计（NetRPC，NSDI23）
- 总结

背景

在网计算对于应用（框架）开发者并不友好，学习曲线陡峭，难以集成到分布式系统中

	应用开发使用的概念	INC中的概念
数据格式	消息	报文
I/O接口	socket、RPC等	packet I/O (DPDK，DMA等）
编程语言	高级语言（C、Java等）	P4硬件语言
资源管理	虚拟化（内存、进程等）	物理资源
运行方式	多应用独立编译，隔离运行	多应用联合编译为单体程序
功能更新	编译改动的单体应用	所有应用集成，重新编译

目标：实现对开发者友好的通信计算库

实现方式

- INC应用分类
- 统一各分类中的数据格式
  - 按照应用开发使用的格式
- 统一交换机中的原语
- 统一应用编程接口
- 系统集成

应用分类（4类）

四个类别：同步聚合、异步聚合、键值读写、共识协议

Table 1: Four Common INC Application Scenarios and Primitives They Need

Type	Applications and Existing Systems	IEDT	Primitives
SyncAgtr	Distributed ML training (ATP [22], SHARP [11], SwitchML [31])	Array	`Map.get`, `Map.addTo`, `Map.clear`, `CntFwd`
AsyncAgtr	MapReduce (ASK [2], NetAccel [23], Cheetah [36])	Map	`Map.get`, `Map.addTo`, `Stream.modify`
KeyValue	Cache (NetCache [19], DistCache [25]), Monitoring (ElasticSketch [38])	Map	`Map.get`, `Map.addTo`
Agreement	Synchronization (P4xos [6], NetChain [18], NetLock [40])	Integer	`Map.get`, `Map.addTo`, `Map.clear`, `CntFwd`

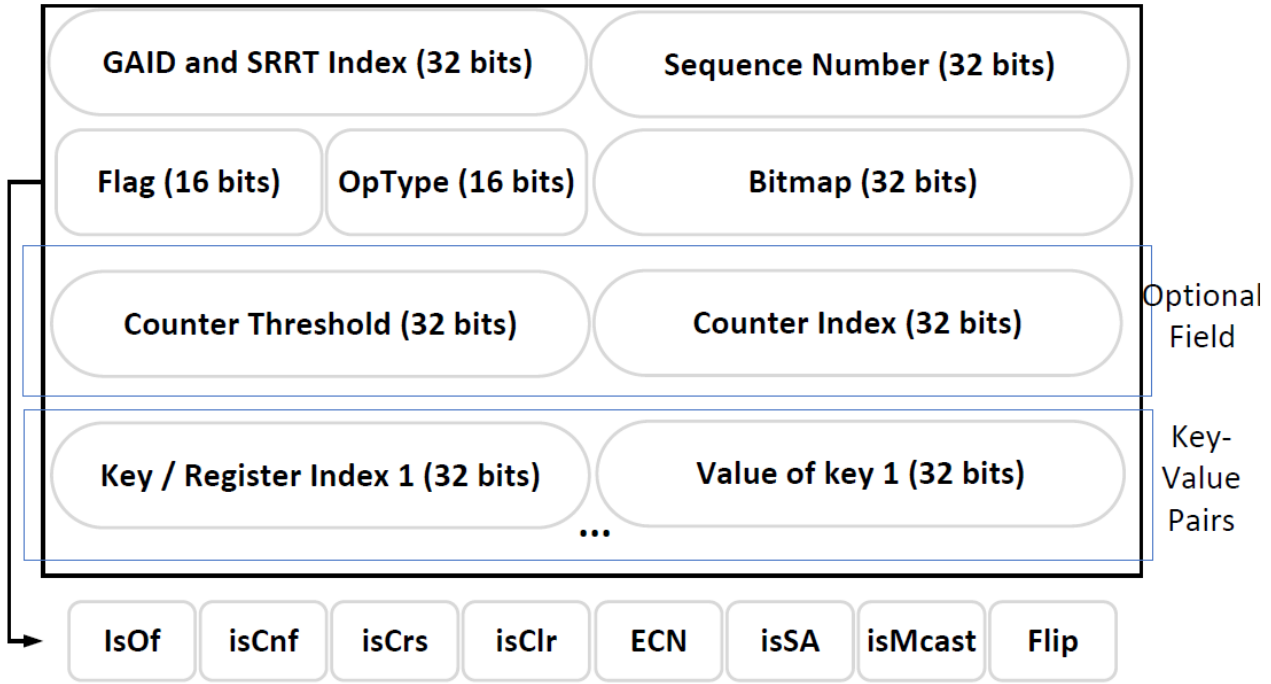
统一数据格式

字典（key-value）、数组（value)、单体变量（int）

Table 1: Four Common INC Application Scenarios and Primitives They Need

Type	Applications and Existing Systems	IEDT	Primitives
SyncAgtr	Distributed ML training (ATP [22], SHARP [11], SwitchML [31])	Array	`Map.get`, `Map.addTo`, `Map.clear`, `CntFwd`
AsyncAgtr	MapReduce (ASK [2], NetAccel [23], Cheetah [36])	Map	`Map.get`, `Map.addTo`, `Stream.modify`
KeyValue	Cache (NetCache [19], DistCache [25]), Monitoring (ElasticSketch [38])	Map	`Map.get`, `Map.addTo`
Agreement	Synchronization (P4xos [6], NetChain [18], NetLock [40])	Integer	`Map.get`, `Map.addTo`, `Map.clear`, `CntFwd`

统一数据包格式

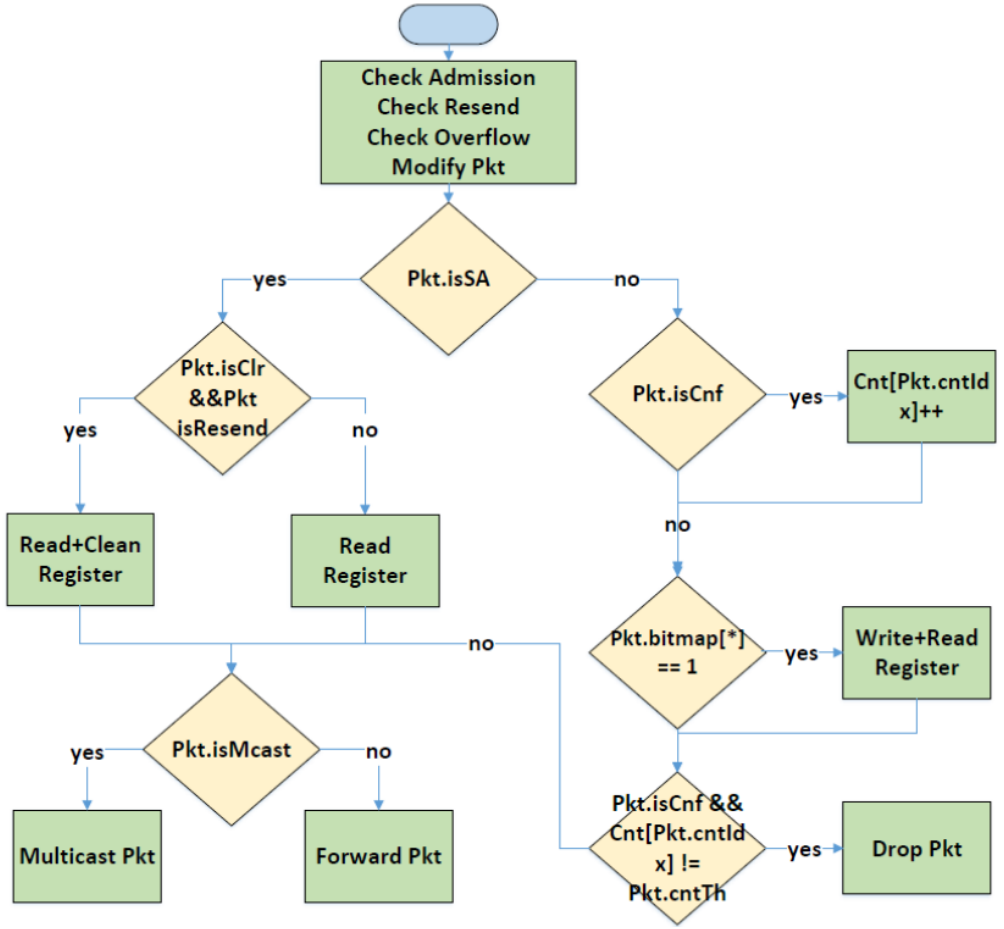

GAID and SRRT Index (32 bits)
Sequence Number (32 bits)
Flag (16 bits)
OpType (16 bits)
Bitmap (32 bits)
Counter Threshold (32 bits)
Counter Index (32 bits)
Optional Field
Key / Register Index 1 (32 bits)
Value of key 1 (32 bits)
...
Key-Value Pairs
IsOf
isCnf
isCrs
isClr
ECN
isSA
isMcast
Flip

统一交换机算子

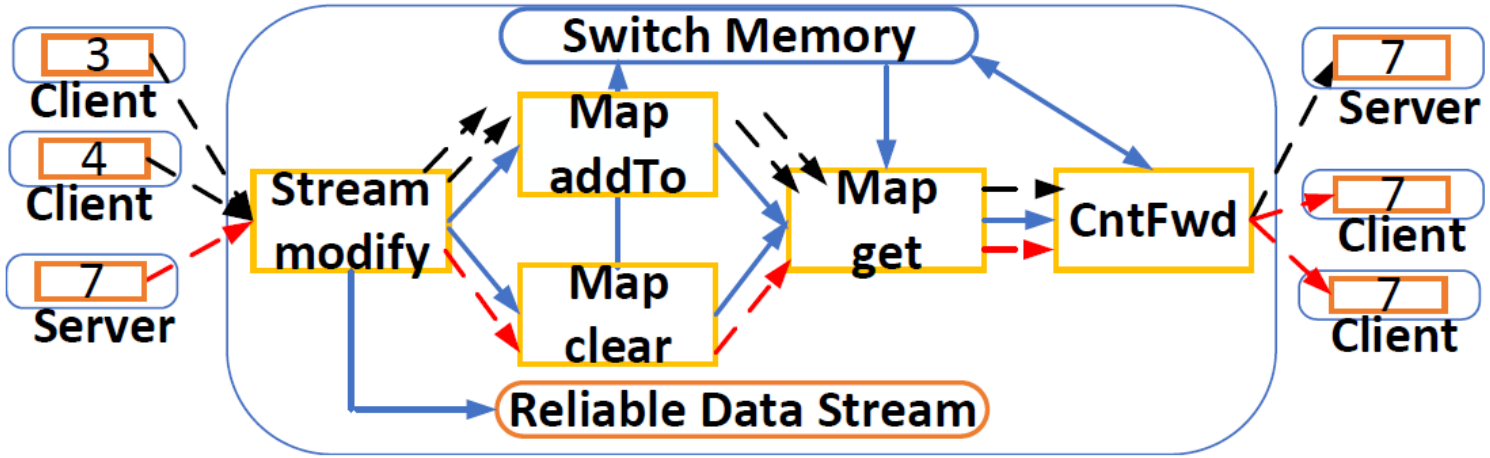
- 五个计算原语
- 在交换机流水线上排布算子
- 通过报文的标志位控制数据通路

Table 2: NetRPC Primitive Semantics

Primitive	Args	Semantics
`Map.addTo`	stream	map[stream.key] += stream.value
`Map.get`	stream	stream.value = map[stream.key]
`Map.clear`	empty	map[stream.key] = 0
`Stream.modify`	op,para	stream.value = op(stream.value, para)
`CntFwd`	key,th,tgt	cnt[key]++; if cnt[key] == th then forward(tgt) else drop

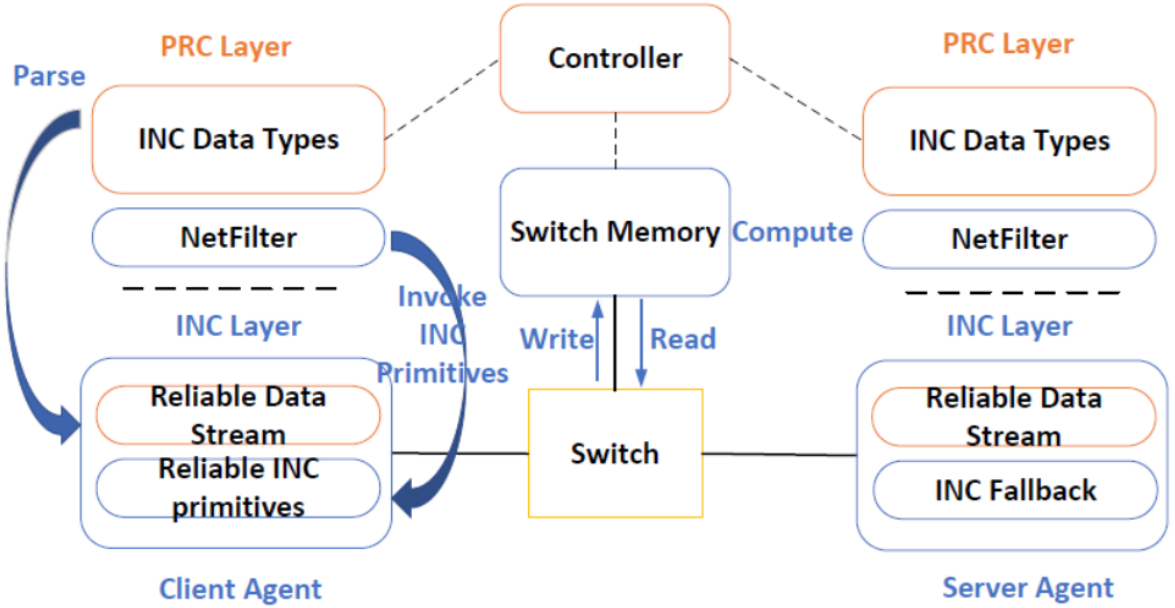


统一交换机算子（ATP为例）



统一编程接口

- 封装为NetRPC数据类型
- 通过配置文件配置原语
- 编译为gRPC接口
  - NetRPC数据走在网计算通道
  - 其他数据走gRPC通道



实验：节约代码行数

Table 3: Workload and Baseline in Experiments

App Type	App	INC Baselines	Dataset
SyncAgtr	Distributed Training	ATP [22] SwitchML [31]	ImageNet [14]
AsyncAgtr	WordCount	ASK [2]	Yelp [39]
KeyValue	Network Monitoring	ElasticSketch [38]	CAIDA Anonymized Internet Trace [4]
Agreement	Paxos	P4xos [6]	Synthatic workload

Table 4: LoC Comparisons: NetRPC vs. Prior INC Arts

	NetRPC		Prior INC Arts	
	Endhost	Switch	Endhost	Switch
SyncAggr	173	13	3394	5329
AsyncAggr	166	26	3278	4258
KeyValue	162	26	898	2360
Agreement	1453	26	5441	931

实验：性能

对比其他INC方案

- 接近的性能

对比端侧系统

- 更高的吞吐量
- 更低的时延

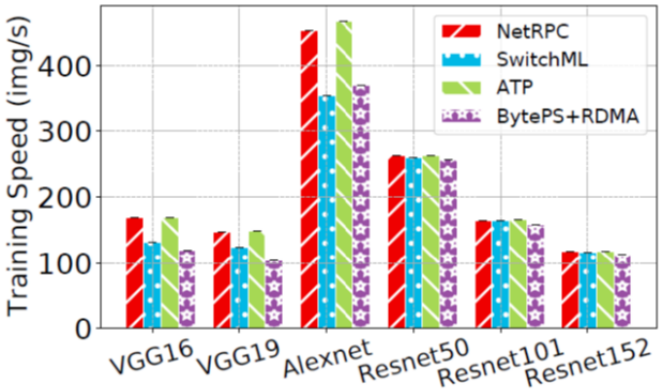
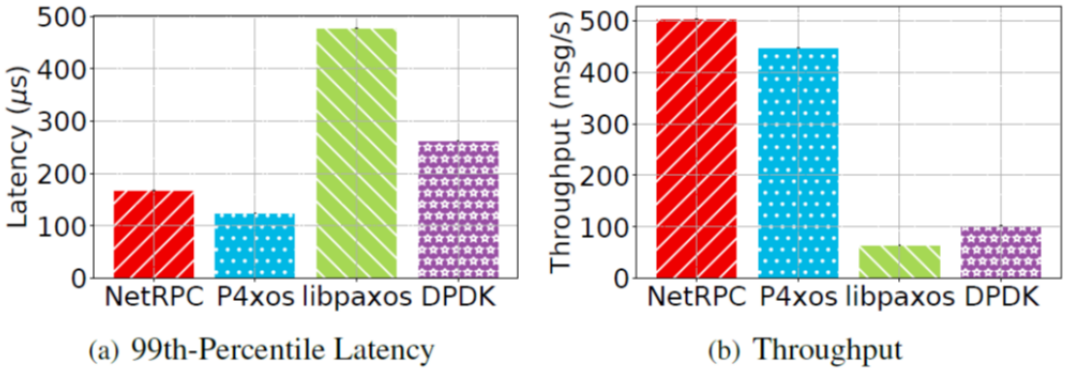


Figure 6: Deep Learning Training Speed



(a) 99th-Percentile Latency (b) Throughput

Figure 7: End-to-end Performance of Paxos Systems.

小结

- 在网计算需要向“应用开发者”提供友好的接口
- 在网计算的多应用需要整合，节约资源
  - 整合交换机逻辑
  - 整合端侧接口

目录

- 在网计算背景和发展路线
- 论文分享
  - 加速分布式机器学习（NetReduce，ASPLOS23；ATP，NSDI21）
  - 分布式机器学习任务管理（INAlloc，INFOCOM23）
  - 加速大数据分析系统（ASK，ASPLOS23）
  - 通信库设计（NetRPC，NSDI23）
- 总结

总结

- 在网计算是有效提升分布式应用效率的一种手段
  - 利用网络设备自带的计算能力，不增加额外设备开销
  - 交换机能效比很高
- 在网计算应用场景广泛，也是一个很大的研究空间
- 我们的实践
  - 两个应用：机器学习、大数据分析
  - 一个编程框架
  - 一个资源管理框架

一些经验

- 在网计算研发难度大
  - 原因：
    - 跨层太多，包括设备、网络层、传输层、通信库、应用功能
    - 评判指标是应用性能
  - 只做一层，难以保证其他层次正确性，难以展示应用改善效果
  - 跨所有层一起开发，周期长，初始门槛高
    - 工业界：不容易找涵盖上述所有部分的团队
    - 学术界：学生培养周期较长
- 希望多交流合作，推动在网计算发展

学生贡献者

论文核心作者

- 何永超：ASK论文第一作者
- 刘俊林：ATP论文共同一作
- 赵伯罕：NetRPC、INAlloc第一作者
- 陈奕熹：ATP参与者、NQ/ATP一作

研究组其他贡献者

- 黄宏毅
- 邓帮文
- 孙铭
- 方少可
- 刘青松

结束

- 谢谢！
- 提问&讨论
- 联系方式
  - 主页：http://wenfei-wu.github.io/
  - 邮箱：wenfeiwu@pku.edu.cn

个人介绍：教育工作经历

北京大学，计算机学院
博士生导师、研究员/助理教授

- 2017-2021，清华大学交叉信息研究院，助理教授
- 2016-2017，美国惠普实验室总部，博士后
- 2010-2015，美国威斯康星大学麦迪逊分校，硕士/博士
- 2006-2010，北京航空航天大学，本科

个人介绍：研究方向

- 信息基础设施的体系结构和运维管理
  - 互联网、云计算、人工智能、大数据、高性能计算
- 方向
  - 在网计算：在传输过程中计算，实现高性能
  - 智能运维：基于AI的监控管理，实现高可靠
  - 开发运维：一体化开发运维，实现高敏捷

个人介绍：研究成果

- 发表论文50篇，CCF A类18篇；中美国专利各3项授权
- 承担科研项目11项
  - 自然科学基金青年项目1项；参与）重点研发计划1项
  - 其他工业界项目若干
- 学术荣誉
  - ASPLOS23杰出论文奖（中国第二次）
  - NSDI21最佳论文（中国首次）
  - IPCCC19最佳论文提名、SoCC13最佳学生论文、SIGCOMM10最佳论文提名