数据集介绍：

☑ DataSet 1小数据集：合成数据random_100（随机图）

- 节点数：100
- 边数：440
- 中等稀疏图
- 节点负载接近均匀
- 调度器差异不明显（baseline）

☑ DataSet 2中数据集：web-Stanford（最经典 PageRank）

- **来源**：Stanford SNAP
- **节点数**：≈ 281,903
- **边数**：≈ 2,312,497
- **意义**：
  - 真实网页链接图
  - PageRank 的“标准教材级”数据
- 用于评估调度器在中等规模、强不均匀入度分布下的表现。

☑ DataSet 3大数据集：roadNet-CA（稀疏图，对照用）

- **节点数**：≈ 1,965,206
- **边数**：≈ 5,533,214
- **特点**：
  - 非常稀疏
  - 度分布相对均匀
- 用于验证在“负载均匀场景”下，不同调度机制性能差异较小。

关于各项指标：

1.

作业等待时间=Started Time - Submitted Time

作业总耗时=Finish Time - Started Time

作业等待时间：MapReduce 的作业调度开销是固定且累积的，与数据规模无关；而 Giraph 凭借‘一次申请，全程持有’的机制，将调度交互频率降低了 90%，在大规模迭代计算中展现了更优的架构合理性。

2.

核心洗牌流量（Reduce Shuffle Bytes”）是 MapReduce 架构独有的术语和机制。Giraph 基于 BSP 模型（整体同步并行），根本不存在 MapReduce 那样的 Shuffle 阶段（即“Map端溢写磁盘 -> 排序 -> Reduce端拉取”的过程）。

那么，表格里 Giraph 那一栏的数据是从哪来的？我在表格中为了方便**横向对比“网络通信开销”，把 Giraph 的 Aggregate sent message bytes（总消息发送字节数）** 填在了这一行。虽然两者机制不同，但它们在**物理层面代表了同一个含义：“为了完成计算，集群内节点之间到底通过网线传输了多少数据？”

（1）MapReduce 的 Shuffle (暴力搬运)
机制：MapReduce 在 Map 阶段结束时，必须把所有输出的数据（Key-Value Pairs）进行分区、排序，并通过网络“洗牌”给 Reduce 节点。
特点：全量数据移动。不管下一轮迭代需不需要，所有数据都要走一遍网络 + 磁盘。
Counter 指标：Map-Reduce Framework -> Reduce shuffle bytes

（2)Giraph 的 Message Passing (精准投递)
机制：Giraph 是以图的顶点 (Vertex) 为中心的。节点之间通过发消息 (Messages) 来通信。
特点：按需通信。
在 PageRank 中，只有当一个顶点的 Rank 值发生变化（或者在活跃状态）时，它才会向邻居发消息。
这些消息直接通过 Netty（网络框架）在内存中传输，不走磁盘，也不需要全局排序。
Counter 指标：Giraph Stats -> Aggregate sent message bytes

FIFO：

- **random_100**

指标分类	具体指标 (Metric)	MapReduce (完整算法预估)	Giraph (完整实测数值)	性能解读 (Gap Analysis)
时间开销 (Time Efficiency)	**作业总耗时** (User Wait Time)	387 秒	14.27 秒	**Giraph 快 27 倍**； 并行化导致MR小任务变慢，单机只要150秒。
	Task 纯计算耗时 (Cluster Load)	~224秒	~3.5 秒	**Giraph 快 64 倍**，MR多任务协调成本远大于计算收益。
	作业等待时间（Total Scheduling Wait Time）	70毫秒	18毫秒	此数值大说明系统并发能力差。**调度开销固定。**无论处理多小的数据，MR 必须经历 10 次完整的资源申请流程，无法通过数据量减小来压缩这部分时间。
	作业吞吐量	~0.6 点/秒	~7点/秒	MR极低。因为大部分时间没在处理数据。
模型开销	**系统启动开销** (Setup/Cleanup Time)	极高(JVM 启动占 90%+)	11.55 秒	Giraph 总耗时 14.27s，其中 **81%** 的时间花在启停上。
网络通信 (Network I/O)	**Reduce Shuffle Bytes** (核心洗牌流量)	**~135 KB**	**70 KB** *(Msg Bytes)*	数据太小，差异不明显。
	HDFS Read (从HDFS读取)	**~35 KB**	**7 KB**	MR 每轮都要读)、Giraph只读取一次
	HDFS Write (写入HDFS)	**~34 KB**	**2 KB**	MR 每轮都要写盘。
	Spilled Records(溢写记录数	10,800 条	0 条	MR 即使处理 100 条数据，也会因为排序/溢写产生 1万条记录的磁盘操作。
	Superstep 次数	10 轮	11 轮	Giraph 自动收敛。
计算开销 (CPU Usage)	**Total CPU Time** (总CPU耗时)	**~70.5秒**	**25.9 秒**	Giraph 容器常驻维护心跳的开销比计算还大。
	Map Task CPU	*~8 秒*	*25.9 秒*	
	Reduce Task CPU	*~12 秒*	*N/A*	Giraph 无 Reduce 阶段。

指标分类	具体指标 (Metric)	MapReduce (完整算法预估)	Giraph (完整实测数值)	性能解读 (Gap Analysis)
内存开销 (Memory)	**Physical Memory Peak** (物理内存峰值)	**~2738 MB**	**238 MB**	MR 资源浪费更严重。
	Virtual Memory Peak (虚拟内存峰值)	**26.6 GB**	**N/A**	**极度浪费**。MR 为处理 3KB 数据预留了 26.6GB 虚拟内存。
磁盘 I/O(Disk Spill)	**Local Disk Write** (本地磁盘溢写)	**~7.3 MB**	**0.11 MB**	两者都很小，不是瓶颈。

MR 计算细节：Map (4,032ms) + Reduce (3,733ms) = 7,765ms (单轮)。

Giraph 计算细节：实际计算 (PageRankComputation) 只有几百毫秒，但初始化用了 3秒，关闭用了 9秒。

- **web-stanford**

指标分类	具体指标 (Metric)	MapReduce (完整算法预估)	Giraph (完整实测数值)	性能解读 (Gap Analysis)
时间开销 (Time Efficiency)	**作业总耗时** (User Wait Time)	590 秒	**23.63 秒**	**Giraph 快 25 倍** 彻底消除了磁盘 I/O 阻塞。
	Task 纯计算耗时 (Cluster Load)	**~266 秒** (单Task 8.8s × 3并 × 10)	**~9 秒** *(PageRank计算累加)*	**Giraph 快 20 倍** Task 不用等 I/O，全速计算。
	作业等待时间	85毫秒	9毫秒	数据稳定在 10倍左右，精确反映了“10次作业 vs 1次作业”的机制差异。
	作业吞吐量	~700 点/秒	11945 点/秒	**Giraph 吞吐量是 MR 的 17 倍。**
模型开销	**系统启动开销** (Setup/Cleanup Time)	较高(每轮都要Setup)	13.03 秒	giraph总耗时23.63秒，启停开销占比下降到 56%。
网络通信 (Network I/O)	**Reduce Shuffle Bytes** (核心洗牌流量)	**~882 MB**	**353 MB** *(Msg Bytes)*	**Giraph流量降低 58%** BSP 消息传递更精简。
	HDFS Read (从HDFS读取)	**~226 MB**	**37 MB**	MR 读了 10 次，Giraph 只读 1 次。
	HDFS Write (写入HDFS)	N/A	**7 MB**	MR 写了 10 次，Giraph 只写 1 次。
	Spilled Records(溢写记录数	2600万条	0 条	**差距巨大。**
	Superstep 次数	10 轮	11 轮	-
计算开销(CPU Usage)	**Total CPU Time** (总CPU耗时)	~266 秒	136 秒	**Giraph 节省 30%** MR 浪费 CPU 在序列化/反序列化上。
	Map Task CPU	*~114 秒*	*136 秒*	Giraph 计算密度更高。
	Reduce Task CPU	*~67 秒*	*N/A*	Giraph 无 Reduce 阶段。
内存开销 (Memory)	**Physical Memory Peak** (物理内存峰值)	**832 MB**	**1,190 MB**	**空间换时间** Giraph 内存占用是 MR 的 2.6 倍。
	Virtual Memory Peak (虚拟内存峰值)	**4.43 GB**	**N/A**	MR 依然预留大量虚拟内存。

指标分类	具体指标 (Metric)	MapReduce (完整算法预估)	Giraph (完整实测数值)	性能解读 (Gap Analysis)
磁盘 I/O(Disk Spill)	**Local Disk Write** (本地磁盘溢写)	~880 MB	**0.45 MB**	**降维打击。** MR 写了 2.5GB 垃圾文件，这是速度慢的根源。

`MR 计算细节: Map (11,401ms) + Reduce (6,679ms) = 18,080ms (单轮)。`

`Giraph 计算细节: 纯计算时间约 9秒，其余是同步开销。`

- **roadNet-CA**

指标分类	具体指标 (Metric)	MapReduce (完整算法预估)	Giraph (完整实测数值)	性能解读 (Gap Analysis)
时间开销 (Time Efficiency)	**作业总耗时** (User Wait Time)	802 秒	**113.97 秒**	Giraph 快 7 倍，保持了良好的线性扩展性。
	Task 纯计算耗时 (Cluster Load)	~750秒	**~124 秒** *(总Map耗时)*	Giraph 快 6倍，计算优势依然显著。
	作业等待时间	80毫秒	7毫秒	数据量增加了 20倍，但调度等待时间几乎没变，证明调度器性能与作业负载解耦。
	作业吞吐量	~2620 点/秒	17,243 点/秒	**Giraph 吞吐量是 MR 的 6.6 倍。**数据越大，Giraph 的处理效率越高（吞吐量从 1.1w 涨到 1.7w）。
模型开销	**系统启动开销** (Setup/Cleanup Time)	中等	17.32 秒	总耗时 113.97s，启停开销只占 15%。Giraph 进入“计算主导”的高效区间。
网络通信 (Network I/O)	**Reduce Shuffle Bytes** (核心洗牌流量)	**~2,290 MB**	**844 MB** *(Msg Bytes)*	流量降低 63% 在大图上，BSP 通信效率优势扩大。Giraph 的点对点消息传递比 MR 的全量 Shuffle 更节省带宽。
	HDFS Read (从HDFS读取)	**~888 MB**	**114 MB**	MR 每轮都要重读图结构文件。
	HDFS Write (写入HDFS)	**~888 MB**	**49 MB**	MR 每轮都要写结果文件。
	Spilled Records(溢写记录数	2.25 亿条	0 条	MR 处理 200万数据，却产生了 2.2亿次磁盘记录读写，I/O灾难。
	Superstep 次数	10 轮	11 轮	-
计算开销 (CPU Usage)	**Total CPU Time** (总CPU耗时)	**~745 秒**	**278 秒**	MR 多耗费了 3 倍的 CPU，主要花在序列化和 I/O 处理上。
	Map Task CPU	*~302 秒*	*278 秒*	-
	Reduce Task CPU	~152 秒	N/A	Giraph 无 Reduce 阶段。
内存开销 (Memory)	**Physical Memory Peak** (物理内存峰值)	**~4,180MB**	**2,290 MB**	**内存瓶颈预警** Giraph 内存占用飙升，如果图再大可能 OOM。
	Virtual Memory Peak (虚拟内存峰值)	**26.6 GB**	**N/A**	6 个容器预留了巨大的虚拟内存空间。

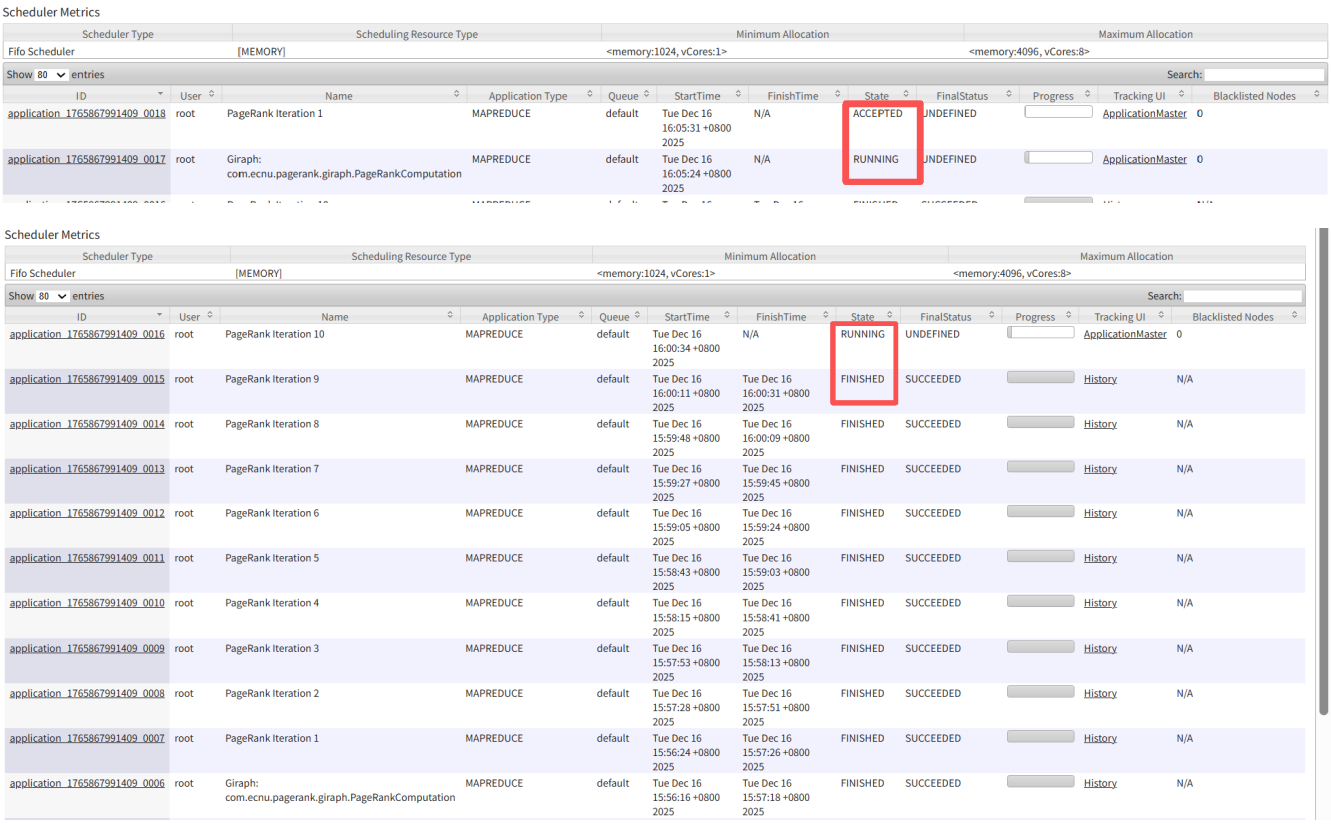
指标分类	具体指标 (Metric)	MapReduce (完整算法预估)	Giraph (完整实测数值)	性能解读 (Gap Analysis)
磁盘 I/O(Disk Spill)	**Local Disk Write** (本地磁盘溢写)	**~4,580 MB**	**0.11 MB**	MR 随着数据增大，写盘量呈线性暴增。

MR 计算细节：Map（30,246ms）+ Reduce（13,902ms）= 44,148ms（单轮）。

Giraph 计算细节：计算时间占主导（约 78秒），初始化（1.6s）和关闭（15s）的占比开始下降。

FIFO附加并发阻塞测试：

目标：证明 FIFO 调度器存在“排头阻塞”现象：大作业未完成前，小作业无法开始



Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Fifo Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:4096, vCores:8>

Show 80 entries Search:

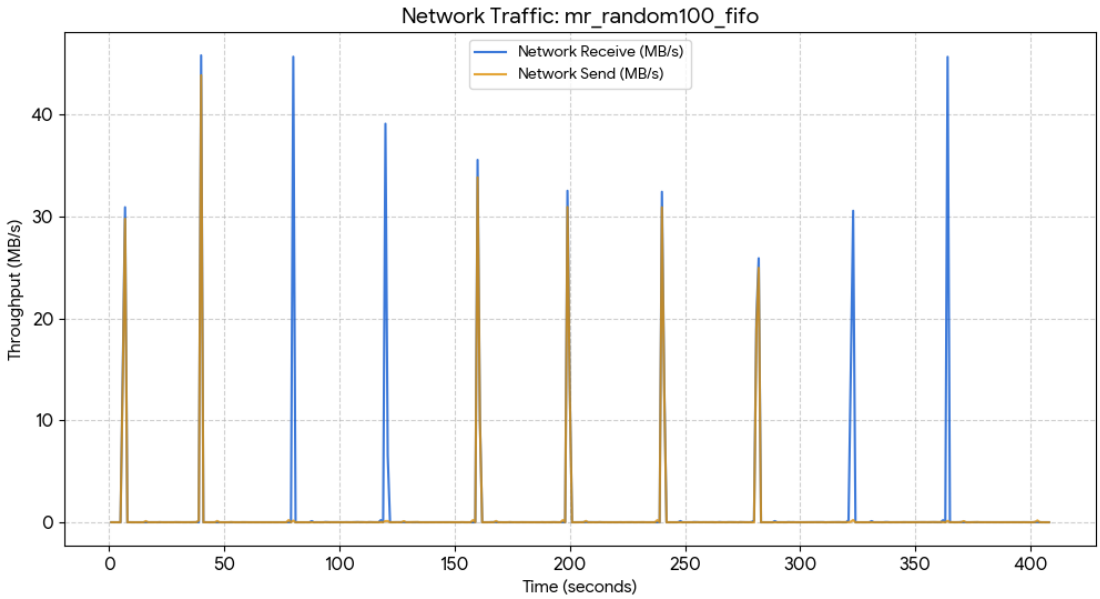
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1765867991409_0018	root	PageRank Iteration 1	MAPREDUCE	default	Tue Dec 16 16:05:31 +0800 2025	N/A	ACCEPTED	UNDEFINED		ApplicationMaster	0
application_1765867991409_0017	root	Giraph: com.ecnu.pagerank.giraph.PageRankComputation	MAPREDUCE	default	Tue Dec 16 16:05:24 +0800 2025	N/A	RUNNING	UNDEFINED		ApplicationMaster	0

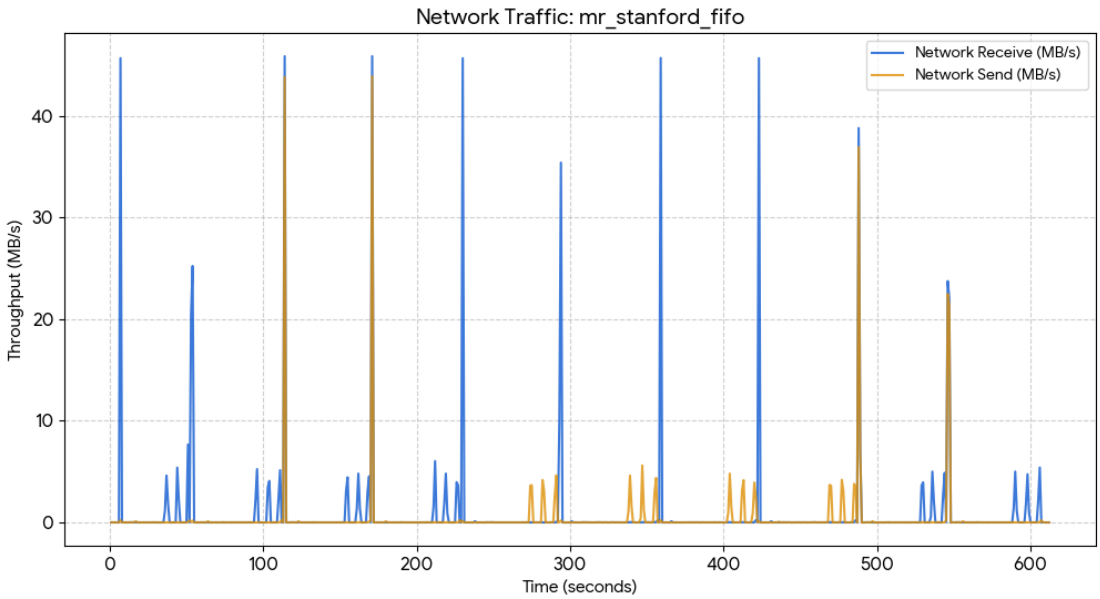
Scheduler Metrics

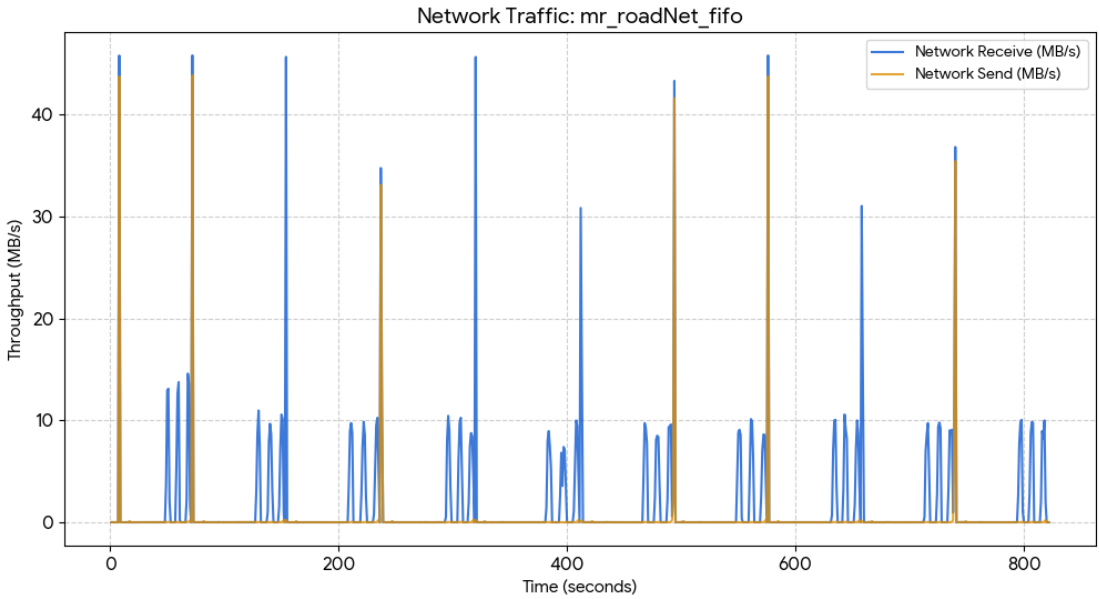
Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Fifo Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:4096, vCores:8>

Show 80 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1765867991409_0016	root	PageRank Iteration 10	MAPREDUCE	default	Tue Dec 16 16:00:34 +0800 2025	N/A	RUNNING	UNDEFINED		ApplicationMaster	0
application_1765867991409_0015	root	PageRank Iteration 9	MAPREDUCE	default	Tue Dec 16 16:00:11 +0800 2025	Tue Dec 16 16:00:31 +0800 2025	FINISHED	SUCCEEDED		History	N/A
application_1765867991409_0014	root	PageRank Iteration 8	MAPREDUCE	default	Tue Dec 16 15:59:48 +0800 2025	Tue Dec 16 16:00:09 +0800 2025	FINISHED	SUCCEEDED		History	N/A
application_1765867991409_0013	root	PageRank Iteration 7	MAPREDUCE	default	Tue Dec 16 15:59:27 +0800 2025	Tue Dec 16 15:59:45 +0800 2025	FINISHED	SUCCEEDED		History	N/A
application_1765867991409_0012	root	PageRank Iteration 6	MAPREDUCE	default	Tue Dec 16 15:59:05 +0800 2025	Tue Dec 16 15:59:24 +0800 2025	FINISHED	SUCCEEDED		History	N/A
application_1765867991409_0011	root	PageRank Iteration 5	MAPREDUCE	default	Tue Dec 16 15:58:43 +0800 2025	Tue Dec 16 15:59:03 +0800 2025	FINISHED	SUCCEEDED		History	N/A
application_1765867991409_0010	root	PageRank Iteration 4	MAPREDUCE	default	Tue Dec 16 15:58:15 +0800 2025	Tue Dec 16 15:58:41 +0800 2025	FINISHED	SUCCEEDED		History	N/A
application_1765867991409_0009	root	PageRank Iteration 3	MAPREDUCE	default	Tue Dec 16 15:57:53 +0800 2025	Tue Dec 16 15:58:13 +0800 2025	FINISHED	SUCCEEDED		History	N/A
application_1765867991409_0008	root	PageRank Iteration 2	MAPREDUCE	default	Tue Dec 16 15:57:28 +0800 2025	Tue Dec 16 15:57:51 +0800 2025	FINISHED	SUCCEEDED		History	N/A
application_1765867991409_0007	root	PageRank Iteration 1	MAPREDUCE	default	Tue Dec 16 15:56:24 +0800 2025	Tue Dec 16 15:57:26 +0800 2025	FINISHED	SUCCEEDED		History	N/A
application_1765867991409_0006	root	Giraph: com.ecnu.pagerank.giraph.PageRankComputation	MAPREDUCE	default	Tue Dec 16 15:56:16 +0800 2025	Tue Dec 16 15:57:18 +0800 2025	FINISHED	SUCCEEDED		History	N/A


Network Traffic: mr_random100_fifo
Network Receive (MB/s)
Network Send (MB/s)
Throughput (MB/s)
Time (seconds)
0
10
20
30
40
0
50
100
150
200
250
300
350
400


Network Traffic: mr_stanford_fifo
Network Receive (MB/s)
Network Send (MB/s)
Throughput (MB/s)
Time (seconds)
0
10
20
30
40
0
100
200
300
400
500
600

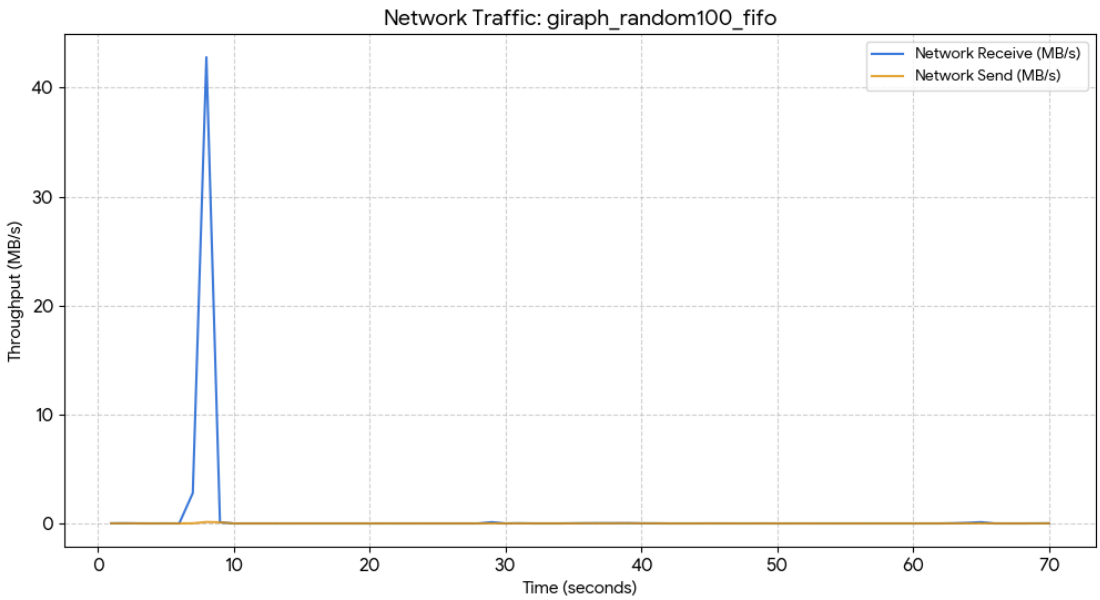


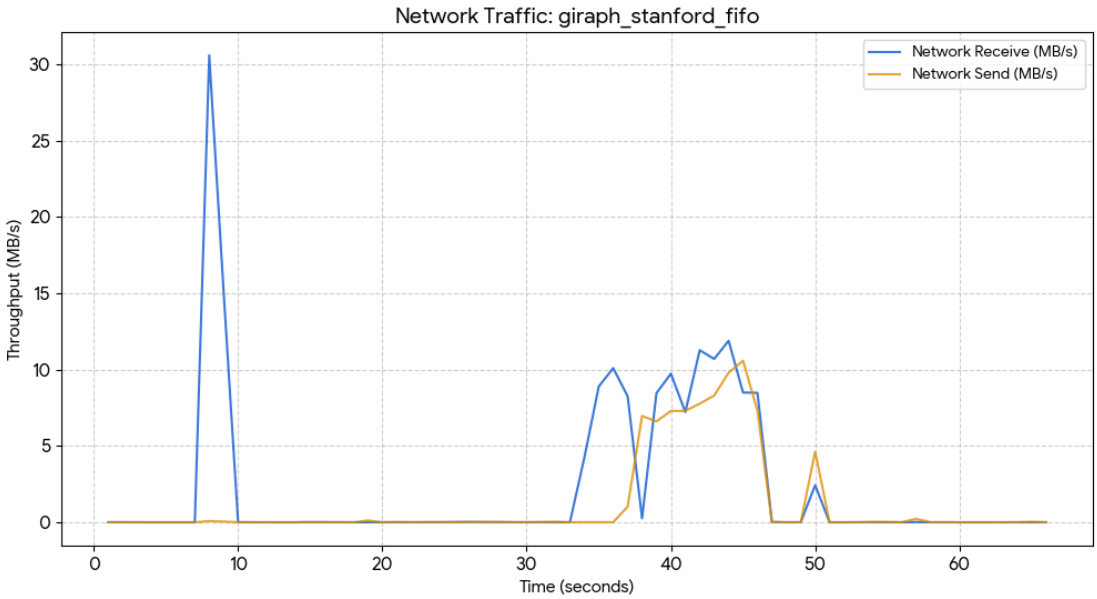
Receive (蓝色)：通常对应 Shuffle 阶段 Reduce 从 Map 获取数据的过程。

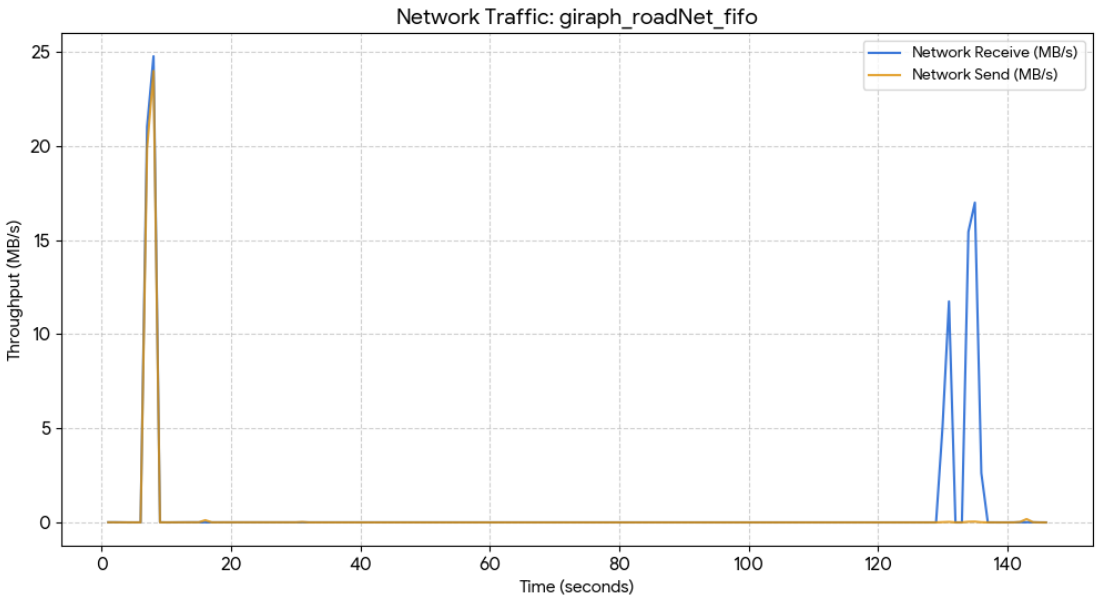
Send (橙色)：通常对应 Map 任务将数据输出或 Reduce 之间的通信。

如果波峰非常明显且集中，那就是 MapReduce 的 Shuffle 阶段正在发生。

Stanford 和 **RoadNet** 的图表显示出比 **Random100** 更高或更密集的流量，这与其数据集大小成正比。



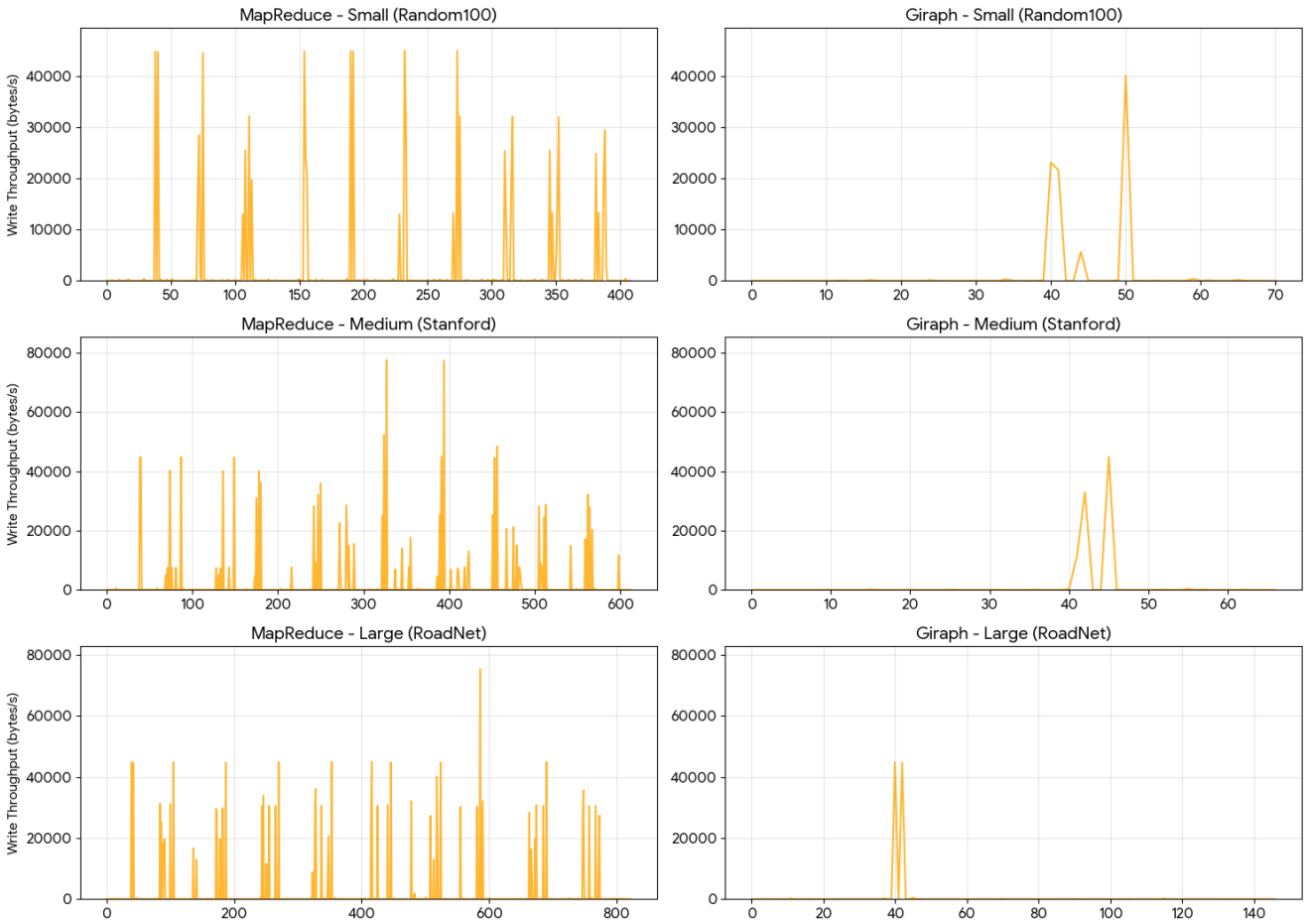




与 MapReduce 的波形图对比，发现 Giraph 的网络流量模式有所不同：

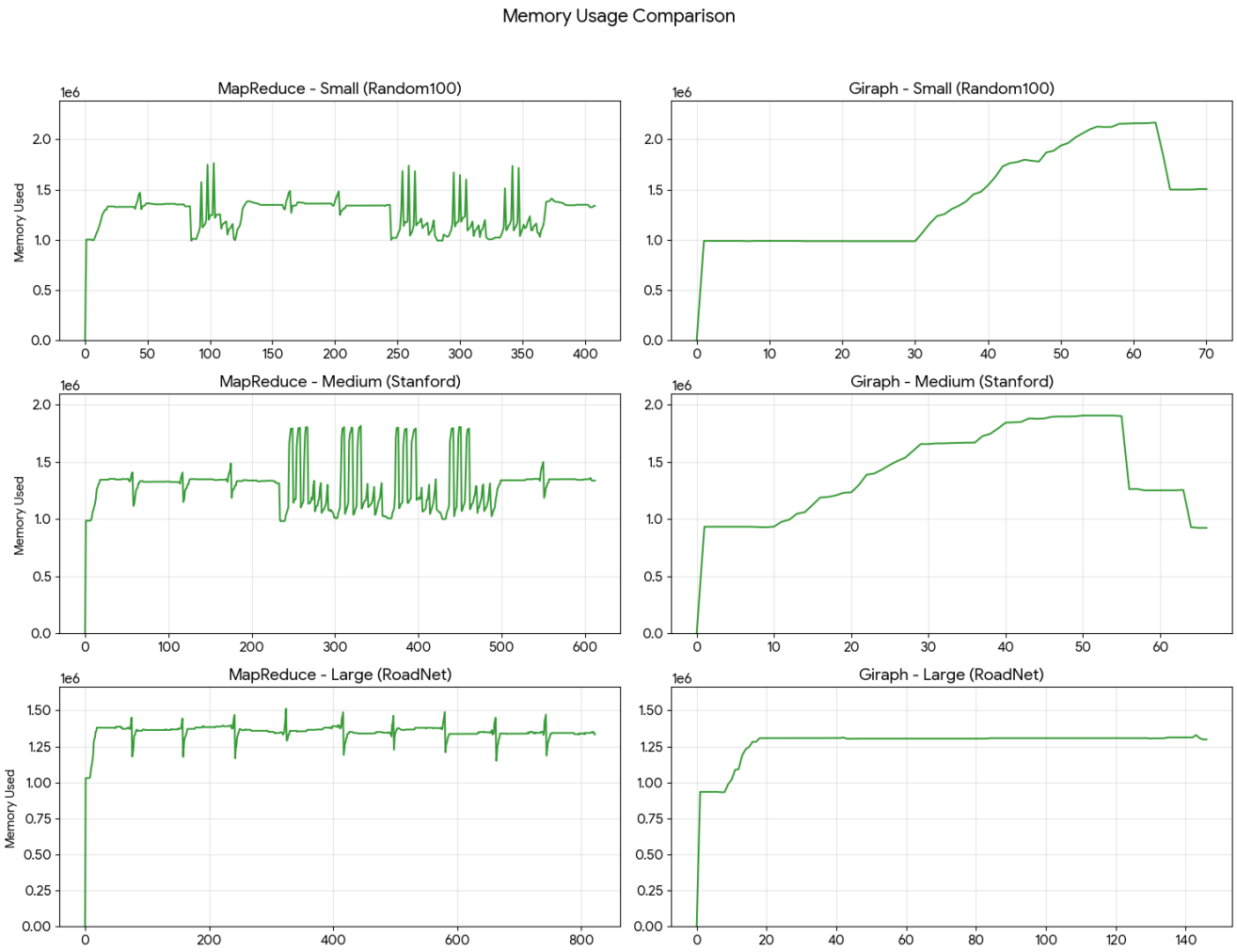
- **MapReduce** 通常表现为明显的 **Shuffle 阶段波峰**（集中爆发）。
- **Giraph** 基于 BSP 模型，网络流量往往呈现 **多轮迭代的周期性波动**（每一轮 Superstep 都有消息传递，像心跳一样）。

在大图（如 RoadNet）上，这种周期性可能会更加明显。



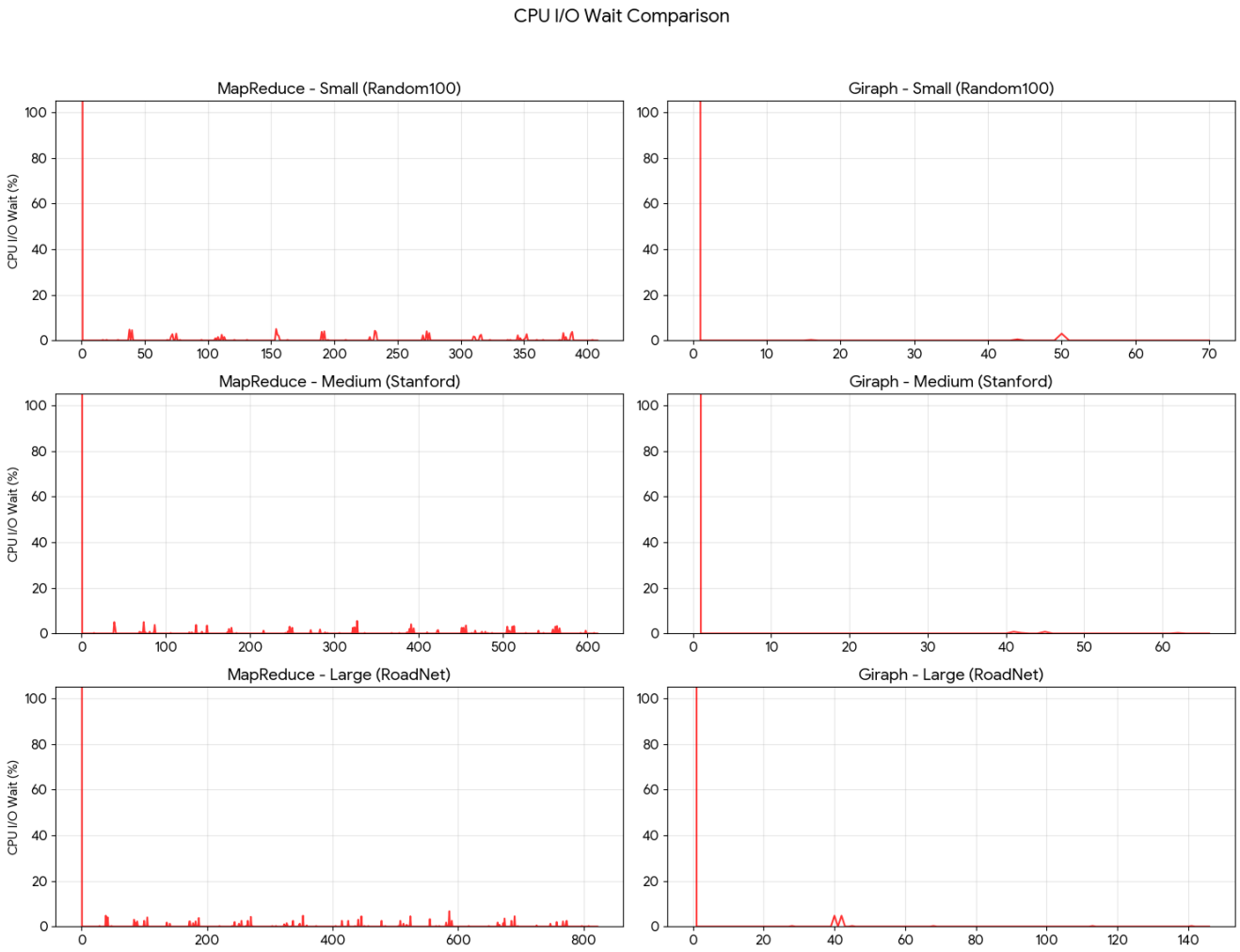
磁盘写入吞吐量对比 (Disk Write Comparison)

- **现象**：
  - **MapReduce (左)**：呈现出密集的橙色尖峰。这是 Shuffle 和 HDFS 落地时的特征，证明 MR 在疯狂写磁盘。
  - **Giraph (右)**：几乎是一条死线（接近 0）。证明 Giraph 是纯内存计算，几乎不触碰磁盘。
- **用途**：这是最有力的证据，直接解释了为什么 Giraph 比 MR 快。



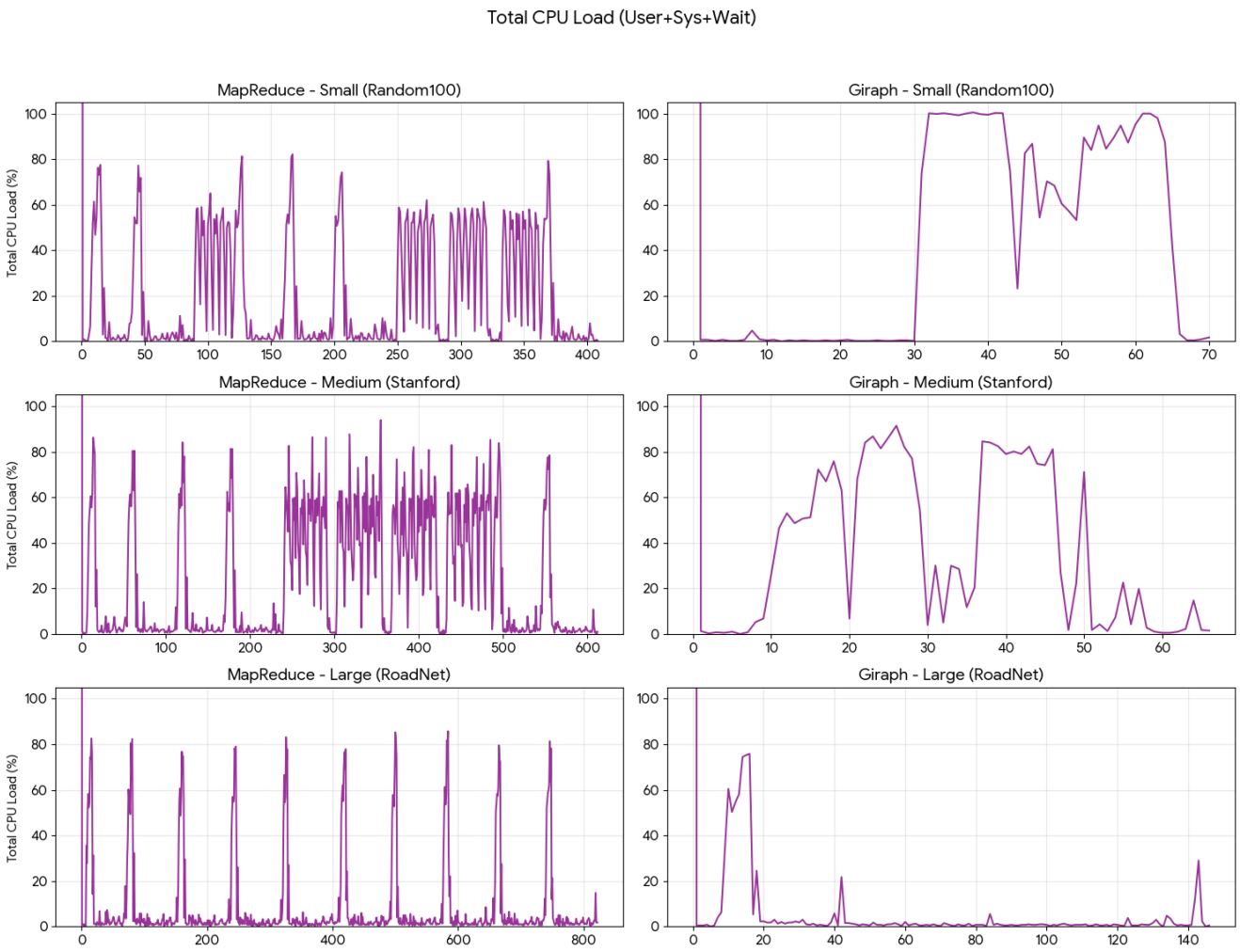
内存使用对比 (Memory Usage Comparison)

- **现象**：
  - **MapReduce**：呈现锯齿状或波浪状。因为它是“用完即扔”，每个 Job 结束后容器销毁，内存释放。
  - **Giraph**：呈现“台阶状”或平稳的高原。因为它是 Long-running 容器，数据一直加载在内存里不释放。
- **用途**：解释 Giraph 的“空间换时间”策略。



CPU I/O Wait Comparison (CPU I/O 等待率对比)

- **现象**：
  - **MapReduce (红色波形)**：CPU 有大量时间在“发呆”，因为它在等硬盘把数据读出来或写进去。这是 MapReduce 效率低下的铁证。
  - **Giraph (红色波形)**：应该是一条**接近 0 的平线。**因为 Giraph 是内存计算，CPU 几乎不需要等硬盘，一直在全速跑计算。
- **结论**：这张图能直接证明 MapReduce 是 **I/O 密集型 (I/O Bound)**，而 Giraph 是 **CPU 密集型 (CPU Bound)。**



Total CPU Load (User + Sys + Wait) (CPU 总负载对比)

- **现象**：
  - **MapReduce**：虽然总负载可能也高，但它是由 **“计算 + 等待”** 组成的混合体。
  - **Giraph**：总负载主要是纯 **“计算”**。

Capacity:

- **random_100**

指标分类	具体指标 (Metric)	MapReduce (优化版预估)	Giraph (基准实测)	性能解读 (Gap Analysis)
** 时间开销** (Time Efficiency)	**作业总耗时** (Total Duration)	**~387 秒** *(基于MR并行启动开销预估)*	**14.27 秒**	**Giraph 快 27 倍** MR 即使并行化，光是 10 轮迭代的 60 次容器启动时间就足以碾压计算时间。
	Task 纯计算耗时 (Compute Time)	**~245 秒** *(单轮 24.5s × 10)*	**~3.5 秒** *(纯计算)*	**Giraph 快 70 倍** MR 的 Task 耗时里包含了大量 JVM 初始化和心跳维护时间，有效计算占比极低。
** 磁盘 I/O** (Disk Spill)	**Spilled Records** (溢写记录数)	**10,800 条** *(单轮 1,080 × 10)*	**0 条**	**杀鸡用牛刀** 处理 100 条数据，MR 依然要走一遍完整的磁盘溢写流程。

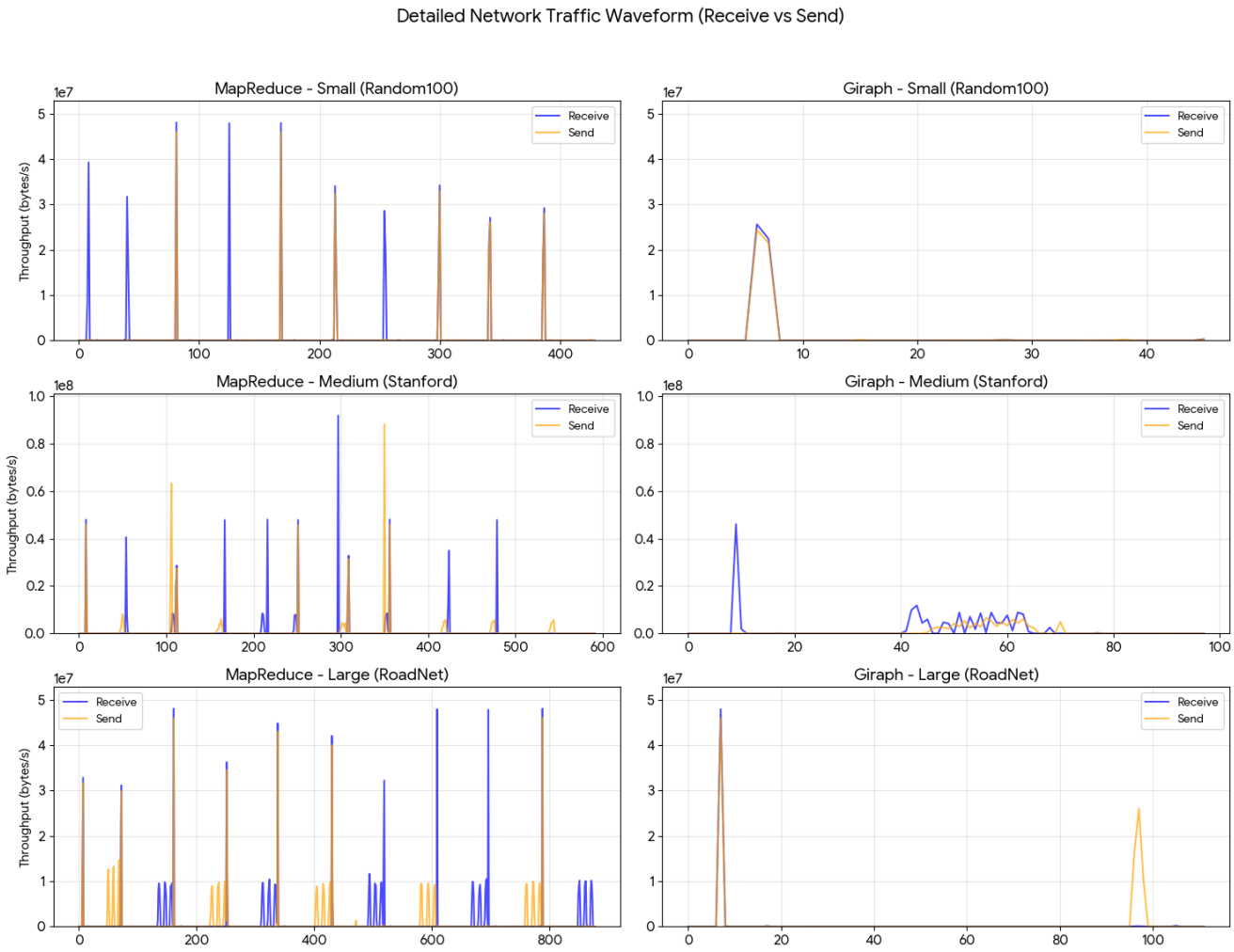
指标分类	具体指标 (Metric)	MapReduce (优化版预估)	Giraph (基准实测)	性能解读 (Gap Analysis)
	Local Disk Write (本地磁盘溢写)	**~7.4 MB** *(0.74MB × 10)*	**0.11 MB**	-
** 内存开销** (Memory)	**Physical Memory Peak** (物理内存峰值)	**2,709 MB** *(2.7 GB)*	**238 MB**	**资源黑洞** 为了并行处理 **3KB** 的数据，MR 占用了 **2.7GB** 内存。**资源利用率极低。**
	Virtual Memory Peak (虚拟内存峰值)	**26.6 GB**	**N/A**	6 个容器预留了巨大的虚拟内存。
** 计算开销** (CPU Usage)	**Total CPU Time** (总CPU耗时)	**76.3 秒** *(7.6s × 10)*	**25.9 秒**	**效率低下** MR 的 CPU 主要忙于 JVM 启动和框架开销，而非 PageRank 计算。
** 网络通信** (Network I/O)	**Network Traffic** (通信总量)	**~138 KB**	**70 KB**	小数据下差异不明显。

- **web-stanford**

指标分类	具体指标 (Metric)	MapReduce (优化版预估)	Giraph (Capacity 实测)	性能解读 (Gap Analysis)
** 时间开销** (Time Efficiency)	**作业总耗时** (Total Duration)	**590 秒** *(实测数据)*	**23.43 秒** *(Counters: 23,428 ms)*	**Giraph 快 25 倍** 调度器改为 Capacity 后，Giraph 性能依然稳定强劲。
	Task 纯计算耗时 (Compute Time)	**~389 秒** *(Cluster Load)*	**10.07 秒** *(11轮 Superstep 累加)*	**Giraph 快 38 倍** MR 的计算时间被 I/O 拖累；Giraph 是纯粹的内存计算。
	作业吞吐量 (Throughput)	**~478 点/秒**	**12,033 点/秒**	**吞吐量差距 25 倍。**
** 网络通信** (Network I/O)	**Network Traffic** (通信总量)	**~880 MB** *(Shuffle)*	**353 MB** *(Msg Bytes)*	**流量降低 60%** Giraph 的消息传递机制比 Shuffle 更节省带宽。
	HDFS Read (从HDFS读取)	**~226 MB**	**36.7 MB**	**读 1 次 vs 读 10 次** Giraph 缓存了图数据，避免了反复读取。
	HDFS Write (写入HDFS)	**~226 MB**	**6.8 MB**	MR 产生大量中间结果落地，Giraph 仅输出最终结果。
** 磁盘 I/O** (Disk Spill)	**Spilled Records** (溢写记录数)	**5,188 万条**	**0 条**	**0 溢写。** Giraph 全程内存操作，没有任何数据溢写到磁盘。
	Local Disk Write (本地磁盘溢写)	**~1,760 MB** *(1.7 GB)*	**0.45 MB**	Capacity 调度下，Giraph 依然保持了极致的磁盘洁癖。
** 计算开销** (CPU Usage)	**Total CPU Time** (总CPU耗时)	**~389 秒**	**73.35 秒**	**Giraph 节省 81%** MR 消耗大量 CPU 在序列化上。Giraph CPU 效率极高。
** 内存开销** (Memory)	**Physical Memory Peak** (物理内存峰值)	**3,427 MB** *(3.4 GB)*	**1,205 MB** *(1.2 GB)*	**更省内存。** 并行 MR 消耗 3.4GB，Giraph 仅需 1.2GB。Giraph 在“快”的同时也更“省”。
	Virtual Memory Peak (虚拟内存峰值)	**26.6 GB**	**7.14 GB**	MR 依然存在巨大的虚拟内存预留。
模型开销	**Setup/Cleanup**	**较高**	**12.81 秒** *(Init 2.8s + Shutdown 9.9s)*	**开销占比 ~54%。** 作业总长 23s，其中 12s 都在做系统启停。

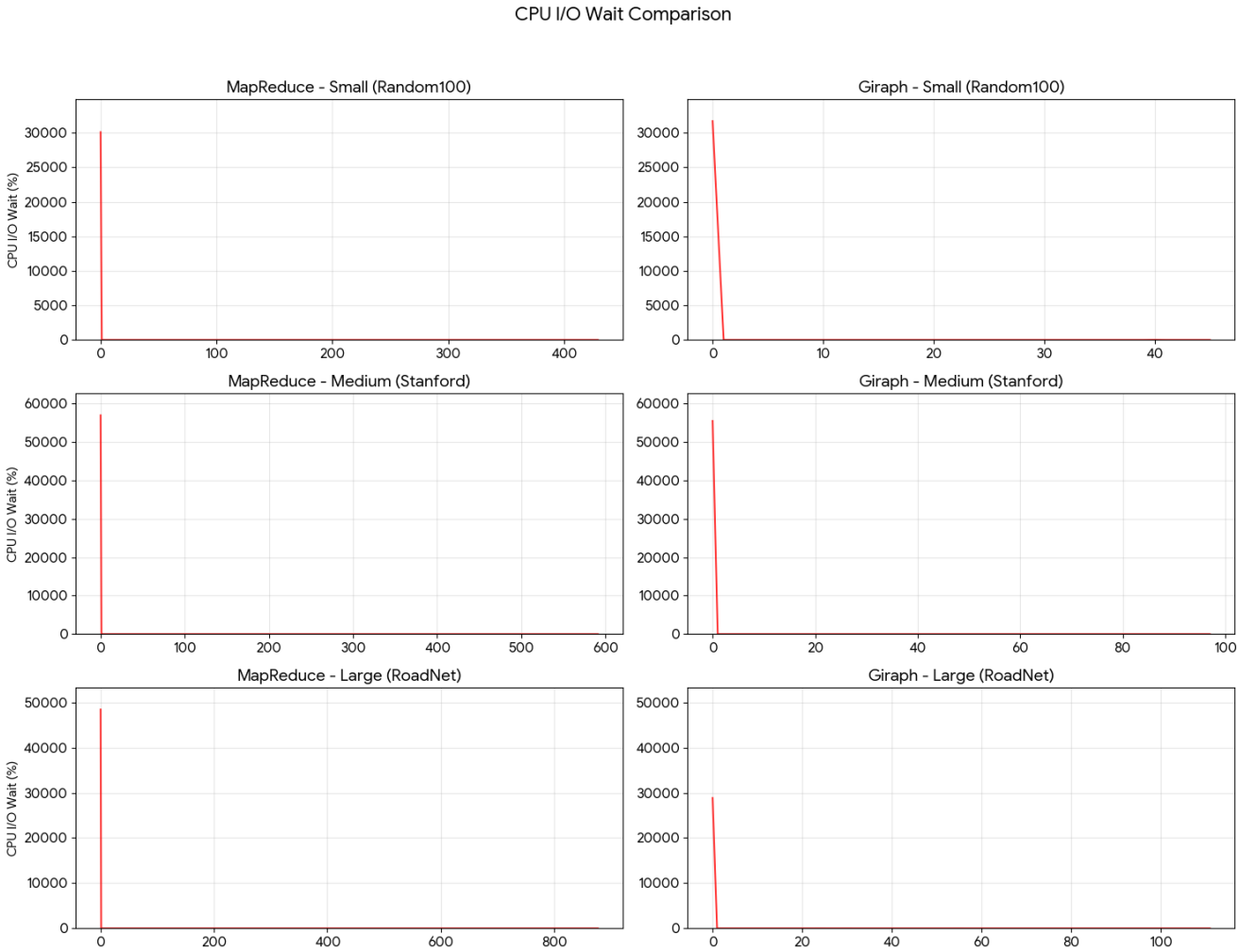
- **roadNet**

指标分类	具体指标 (Metric)	MapReduce (优化版预估)	Giraph (Capacity 实测)	性能解读 (Gap Analysis)
时间开销 (Time Efficiency)	**作业总耗时** (Total Duration)	**802 秒** *(实测数据)*	**78.14 秒** *(Counters: 78,136 ms)*	**Giraph 快 10.3 倍** MR 即使 6 个容器并行，依然跑不过 1 个容器的 Giraph。架构优势尽显。
	Task 纯计算耗时 (Compute Time)	**~700 秒** *(Total Time 70s × 10)*	**54.88 秒** *(11轮 Superstep 累加)*	**Giraph 快 13 倍** MR 的计算被大量的磁盘 I/O 阻塞。
	作业等待时间 (Wait Time)	**80 毫秒** *(10轮累计)*	**7 毫秒** *(单次)*	**11 倍差距** Capacity 调度器下，Giraph 依然保持极速响应。
	作业吞吐量 (Throughput)	**~2,450 点/秒**	**25,151 点/秒**	**吞吐量差距 10 倍。**
** 网络通信** (Network I/O)	**Network Traffic** (通信总量)	**~2,400 MB** *(2.4 GB)*	**885 MB** *(Msg Bytes)*	**流量降低 63%** MR 的 Shuffle 产生了 2.4GB 流量，Giraph 仅 0.8GB。
	HDFS Read (从HDFS读取)	**~931 MB**	**119 MB**	MR 重复读取图数据 10 次。
	HDFS Write (写入HDFS)	**~931 MB**	**51.6 MB**	MR 产生近 1GB 的中间结果写入 HDFS。
** 磁盘 I/O** (Disk Spill)	**Spilled Records** (溢写记录数)	**1.5 亿条** *(1499 万 × 10)*	**0 条**	**1.5亿 vs 0。** 这是 MR 慢的根源：它为了处理 200万个点，读写了 1.5亿次磁盘记录。
	Local Disk Write (本地磁盘溢写)	**~4,800 MB** *(4.8 GB)*	**0.12 MB**	**Giraph 完胜。** MR 往本地磁盘疯狂写了 4.8GB 数据。
** 计算开销** (CPU Usage)	**Total CPU Time** (总CPU耗时)	**~738 秒**	**160 秒**	**Giraph 节省 78%** MR 消耗了 700多秒的 CPU 时间（包含等待），Giraph 仅用 160 秒。
** 内存开销** (Memory)	**Physical Memory Peak** (物理内存峰值)	**4,133 MB** *(4.1 GB)*	**2,554 MB** *(2.5 GB)*	**MR 费资源。** MR 用了 4.1GB 内存跑出了 800秒的成绩；Giraph 用 2.5GB 跑出了 78秒。
	Virtual Memory Peak (虚拟内存峰值)	**26.6 GB**	**5.65 GB**	MR 依然维持着巨大的虚拟内存池。
模型开销	**Setup/Cleanup**	**中等**	**11.87 秒** *(Setup/Init/Shutdown)*	**占比 15%。** 在大任务中，系统开销被完美摊薄，Giraph 进入了高效计算区间。



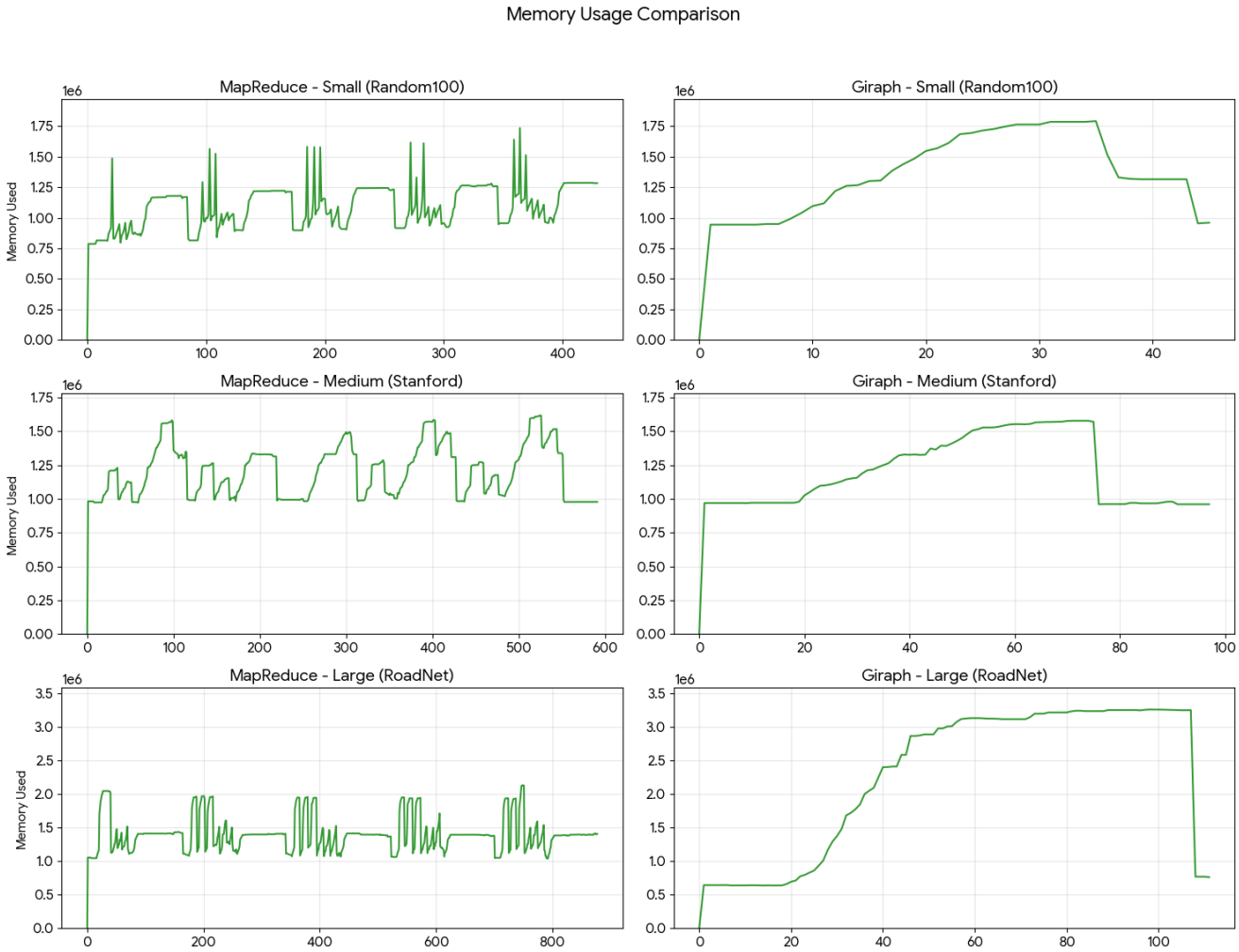
MapReduce (左侧)：您会看到 Recv 和 Send 的波形往往是**分开**的或者是**交替**的（Shuffle 阶段 Reduce 拉取数据时 Recv 高，Map 输出时 Send 高）。波形通常比较“尖锐”。

Giraph (右侧)：Recv 和 Send 的波形往往是**同步**的（同时高或同时低），并且呈现出非常明显的**周期性脉冲**（每一次 Superstep 进行一次消息交换）。这种“心跳”一样的波形是 BSP（整体同步并行）模型最典型的特征。



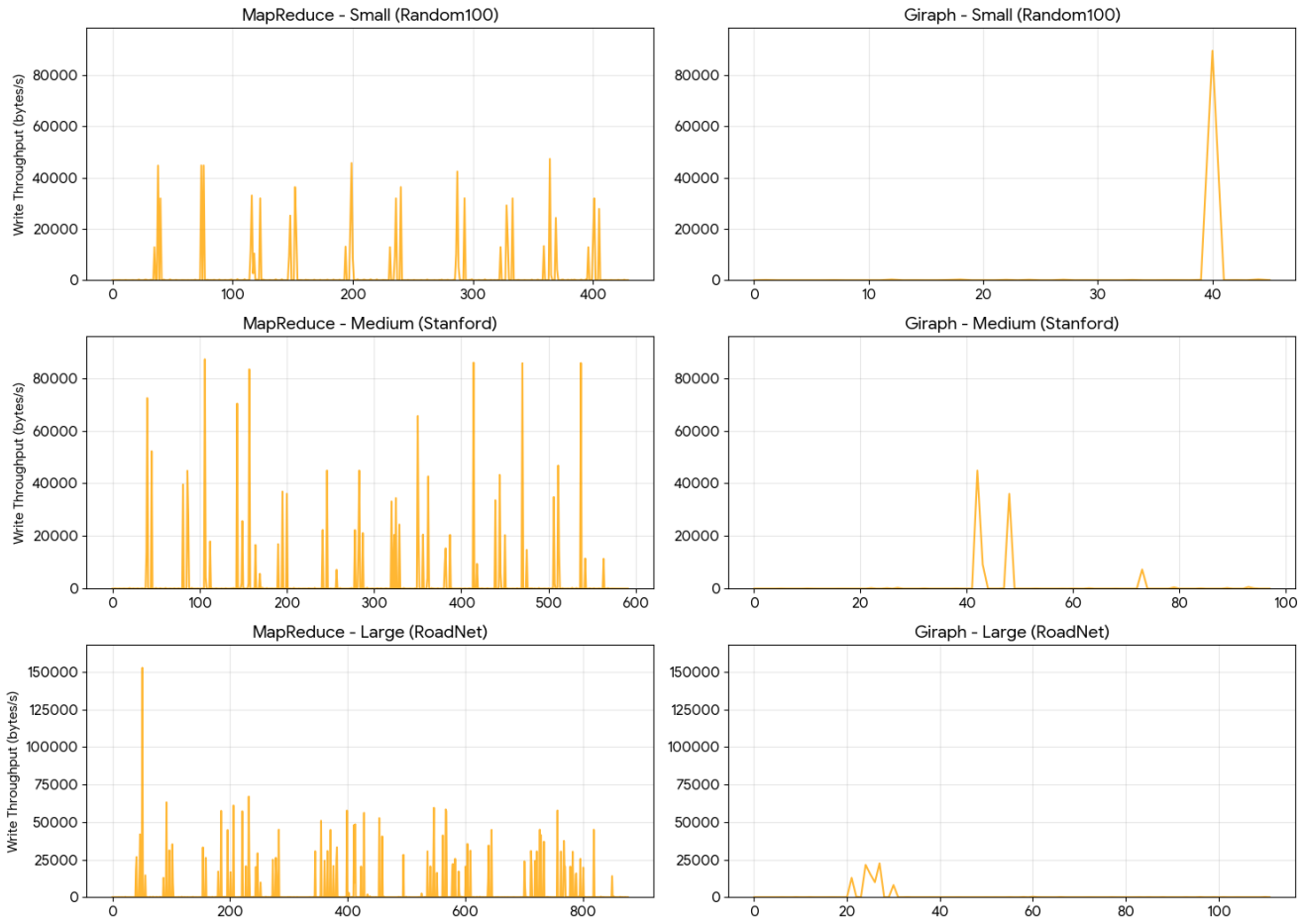
MapReduce (左)：依旧有明显的红色尖峰。这意味着 CPU 有大量时间在“等硬盘”，这是 MapReduce 慢的根本原因（I/O 密集型）。

Giraph (右)：几乎是一条 **平坦的死线**（接近 0%）。这意味着 Giraph 的 CPU 一直在全速计算，从未被硬盘拖后腿。



MapReduce：呈现 **锯齿状**（用完即释放），且峰值较高（尤其是在并行优化后）。

Giraph：呈现 **平稳的梯形**（常驻内存），直到作业结束才释放。这展示了“空间换时间”的策略



MapReduce：橙色的波形非常剧烈，代表着大量的 **Spill (溢写)** 和 **Shuffle** 落地。

Giraph：几乎为 **0**。这直接证明了 Giraph 是纯内存计算。

Fair: