测试与分析

1. 物理实验环境

本实验的实验环境是真实的物理集群，集群规模是 10 台真实物理机器，其中 9 台节点做 slave 节点，一台节点作为 master 节点。节点配置信息如下表 1 所示：

节点基本信息	
处理器	8 Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
内存	16GB RAM
操作系统	Ubuntu 11.04
网络带宽	1000MBps
磁盘	2* 1TB SATA
Spark version	Spark 2.0

表 **1** 节点信息

2 测试数据集

本文测试来自于真实的问题和数据集，在算法上选择 ALS 算法作为实验的算法，同时选择了 MovieLens [17,18]和 Netflix[19]作为 ALS 算法测试数据集。算法和数据集的详细描述如下表所示：测试数据集如表 2 所示：

数据集	数据集说明
MovieLens	MovieLens 数据集是由美国 Minnesota 大学的 GroupLens 研究小组创建并维护，其研究项目涉及相关的许多研究项目到信息过滤，协同过滤等领域推荐系统。本数据集是 GroupLens 发布的数据集，它包括 260,000 个用户在 4 万部电影中应用了 24,000,000 个评分和 670,000 个标签应用。所有的评分值也都是 1 到 5 中的整数值，其中分数越高表示客户对相应电影的评价越高（越喜欢）
Netflix	Netflix 数据集是由 Netflix-prize 比赛提供的公开数据集，根据历史用户对电影的评价文件对用户进行电影推荐，历史电影评价文件包含超过 1 亿的评分从 48 万随机选择，匿名 Netflix 客户超过 17 万部电影的数据收集于 1998 年 10 月至 2005 年 12 月，并反映了在此期间收到的所有评级的分

	布。 评分是在规模从 1 到 5 星。 为了保护客户的隐私，每个客户 id 已替换为随机分配的 ID。每个评级的日期和还提供每个电影 id 的发行的标题和年份。所有的评分值都是 1 到 5 中的整数值，其中分数越高表示客户对相应电影的评价越高（越喜欢）

表 **2** 算法测试数据集

3 观测指标

本文采用了[7-10]文献中的观察指标，所不同的是，本文提出了更加细粒度的时间指标 ,结合实验观察 ,任务执行的时间包括数据处理时间和系统开销，在进行实验时，对任务执行的时间进行细化，分别测定数据处理时间和系统开销。

4 实验结果分析

在 Netflix 数据集上，每个 executor 设置 4GB 内存，包含两个 core，相同任务并行度下，使 core（CPU）个数逐渐增大，测试结果如图 1 所示，

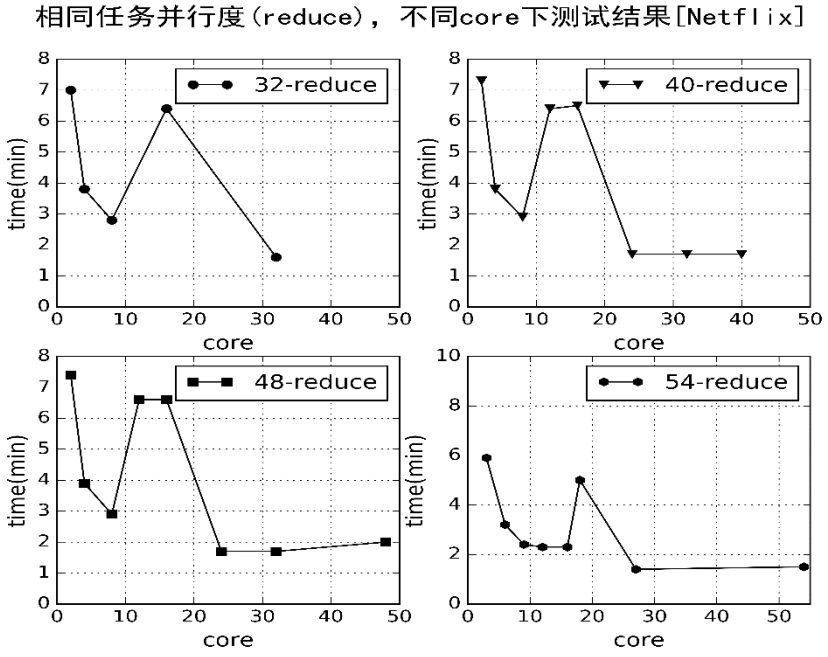

图 **1** Netflix 上横向扩展物理资源测试结果

随着 core 个数的增大，算法的执行时间先减小，突然异常增大，之后当 core 的个数超过某个值时，执行时间再减小。随着物理资源 core（CPU）的增大，出现了执行时间增大的异常情况。在异常点进一步细化物理资源 core 的增

长幅度，分别在 32 个任务并行度和 48 个任务并行度下，逐渐增加物理资源 core（CPU），实验结果如图 2 所示，

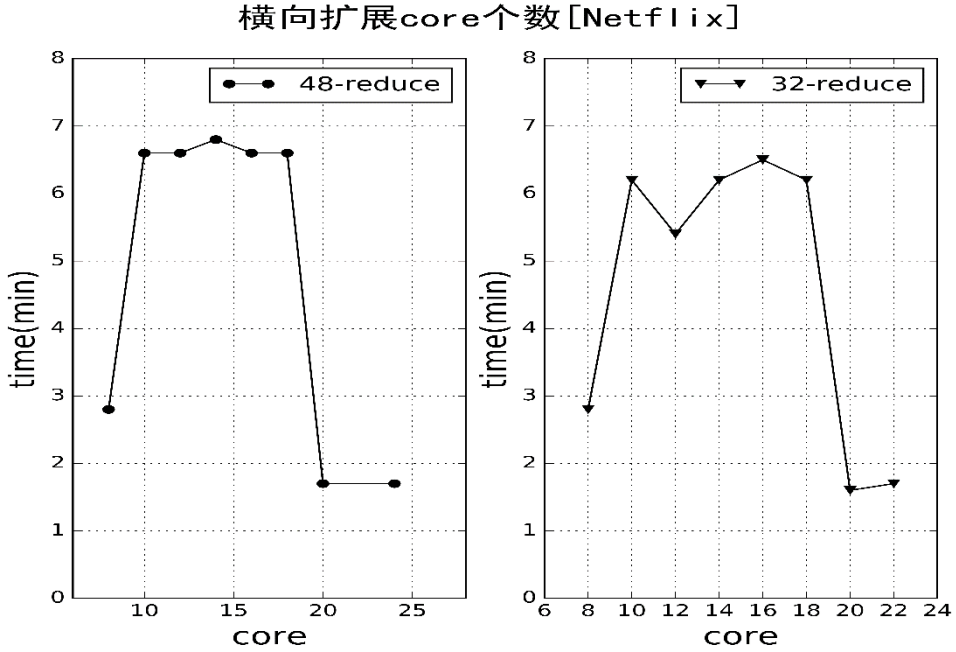

图 **2**　Netflix 上异常点横向扩展物理资源测试结果

从图 2 中可以看出 core 的个数在 10 到 18 之间时，执行时间超过了 6 个 core 的执行时间，通过对 Spark UI 中的监控信息和执行流程图进行分析，发现如图 3 所示。

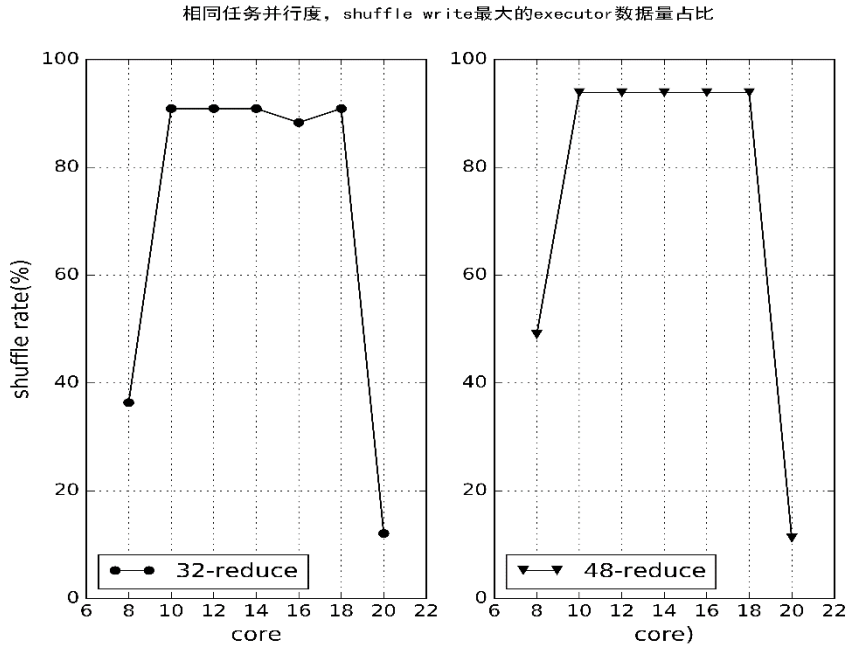

图 **3** Netflix 上最大的 executor shuffle 数据占总任务 shuffle 数据的比率

存在某个 executors 的 shuffle 数据量占到了总任务 shuffle 数据量的 90%以上，造成大量的任务在 executor 上执行，这是由于 RDD 的优先位置(Preferred

location)属性在计算每一个分片的优先计算位置造成大量中间结果聚集在某个节点(executor)上，为了验证这个分析，改变了 executor 中内存的大小和 executor 在节点中的分布，但仍保证每个 core 的内存大小是 2 GB，执行结果如下图所示，

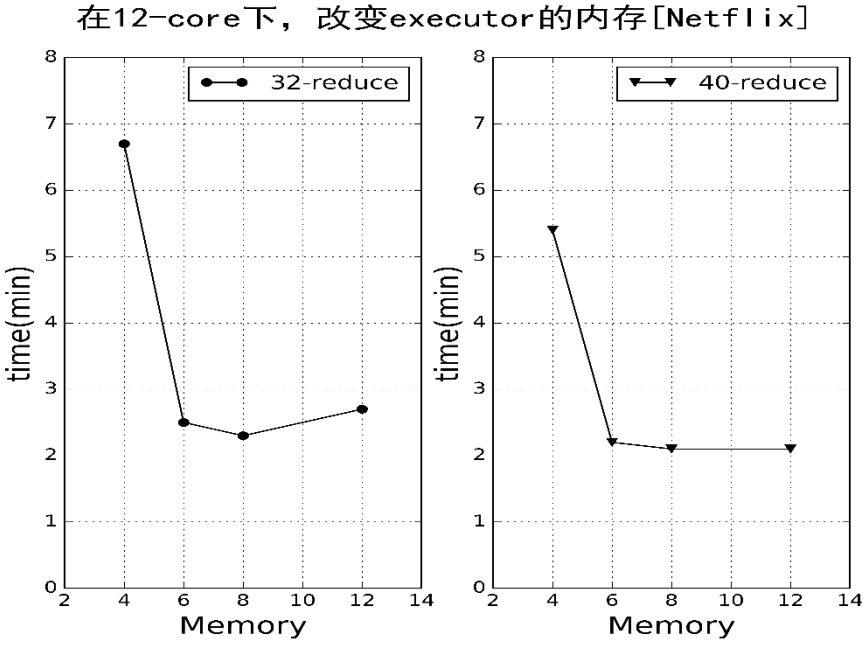

图 **4**　Netflix 上改变 executor 的分布后测试结果

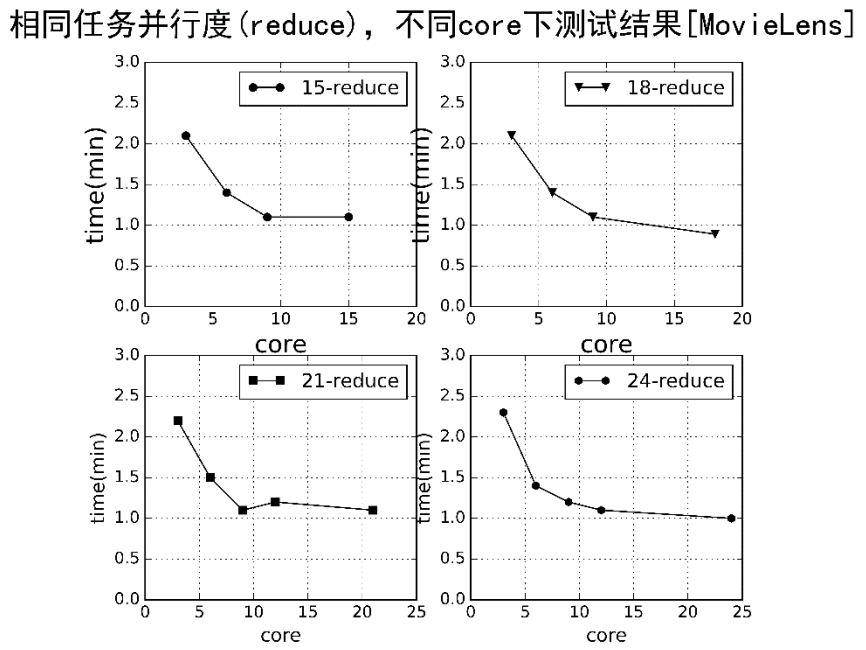

图 **5**　MoviesLens 横向扩展物理资源的测试结果

从图 1、图 4 和图 5 中可以看出，在确定任务并行度时，随着物理资源的增加，执行时间最后接近于趋向于一个不变的值，即对着物理资源的增加，系统的执行时间不是线性下降的。

在 Netflix 数据集上，每个 executor 设置 4GB 内存，包含两个 core（CPU），相同物理资源 core，不同任务并行度下的执行结果如图 6 所示，

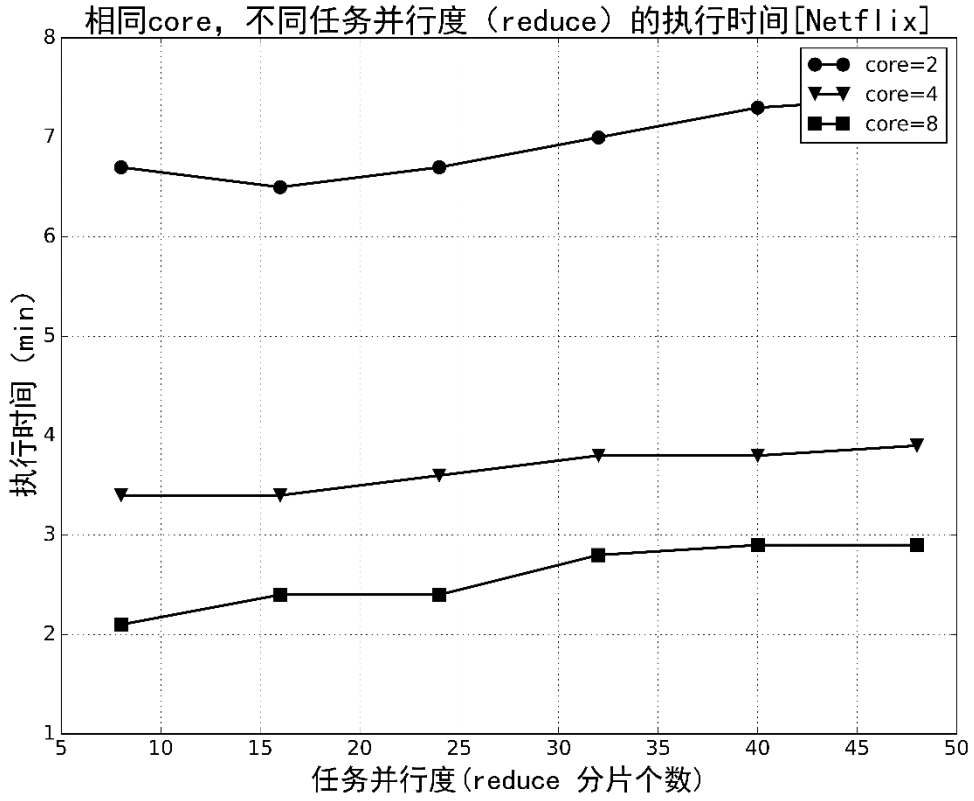

图 **6**　MoviesLens 横向扩展物理资源的测试结果

在 MovieLens 数据集上每个 executor 设置 4GB 内存，包含两个 core（CPU），相同物理资源 core，不同任务并行度下的执行结果如图 7 所示，

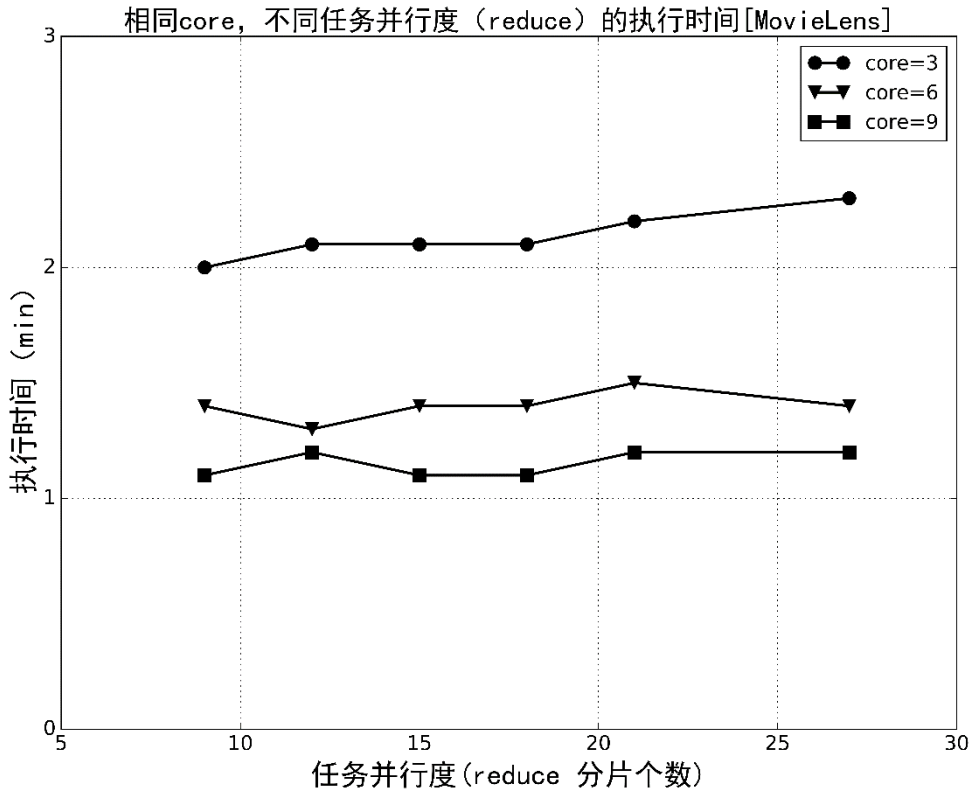

图 **7**　MoviesLens 横向扩展物理资源的测试结果

从图 6 和图 7 可以看到，在相同物理资源的条件下，随着任务并行度的增加，执行时间是一直增大的，这是因为在 ALS 算法在实现中，把原始评价矩阵分解成不同的子矩阵，每个子矩阵相互独立，但是在进行迭代计算时，下一轮计算需要从上一轮的子矩阵中获取数据，如图 8 所示，左图中 User1，User2，User3 分别位于不同的分区中，右图中 User1，User2 位于同一个分区中。

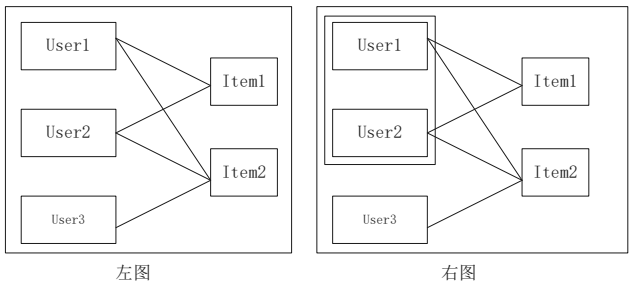

图 **8** MoviesLens 横向扩展物理资源的测试结果

求解 Item1 需要获得 User1,User2，求解 v2 需要获得 User1,User2，User3 等，如果计算 User1 和 User2 是在同一个分区上进行的(右图) ,那么只需要把 User1 和 User2 一次发给这个分区，而不需要将 User1,User2 分别发给 Item1,Item2，这样就省掉了不必要的数据传输，减少了计算开销。算法的任务并行度越高，分区的大小越小，那么在计算中需要进行的传输开销就越大，所以在算法使用中，应使用大小适中的任务并行度，较小数据传输开销，提高系统效率。