中国科学技术大学
University of Science and Technology of China

面向多核可伸缩的MapReduce库的研究

报告人：俞玉芬

导　师：张　昱

内容概要

01 研究背景与动机

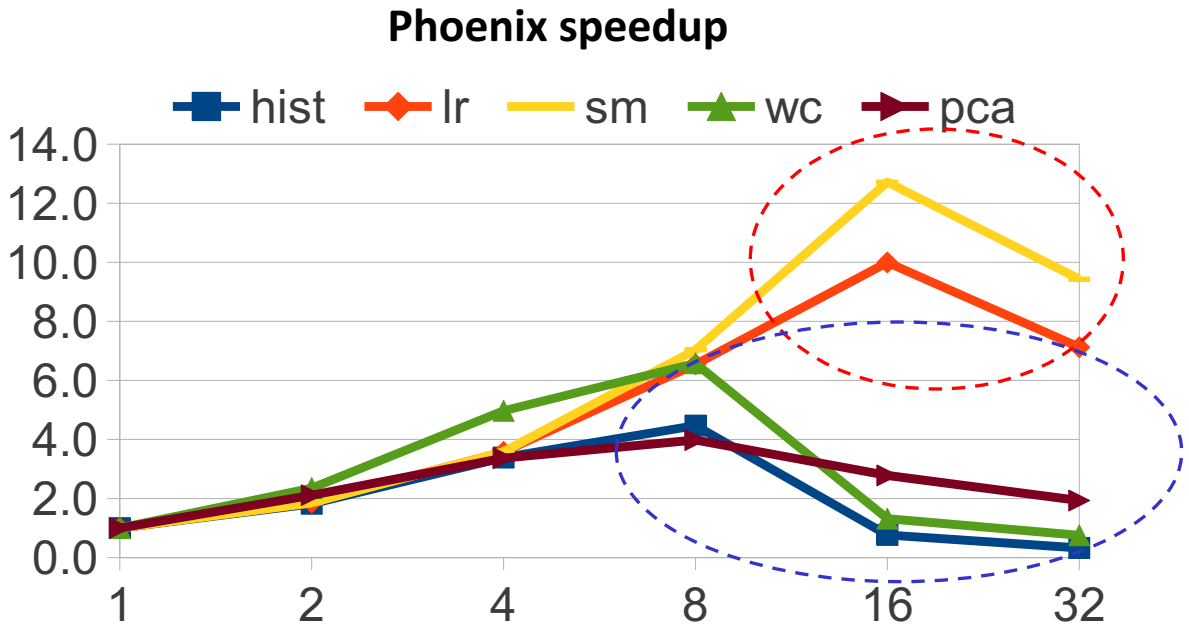
02 SMR的总体设计

03 实验结果与分析

研究背景——多核MapReduce的相关研究

多核机器的广泛普及，如何充分利用多核资源？**并行编程**

面向多核MapReduce库——Phoenix



hist – histgram

lr – linear_regression

sm—stringmatch

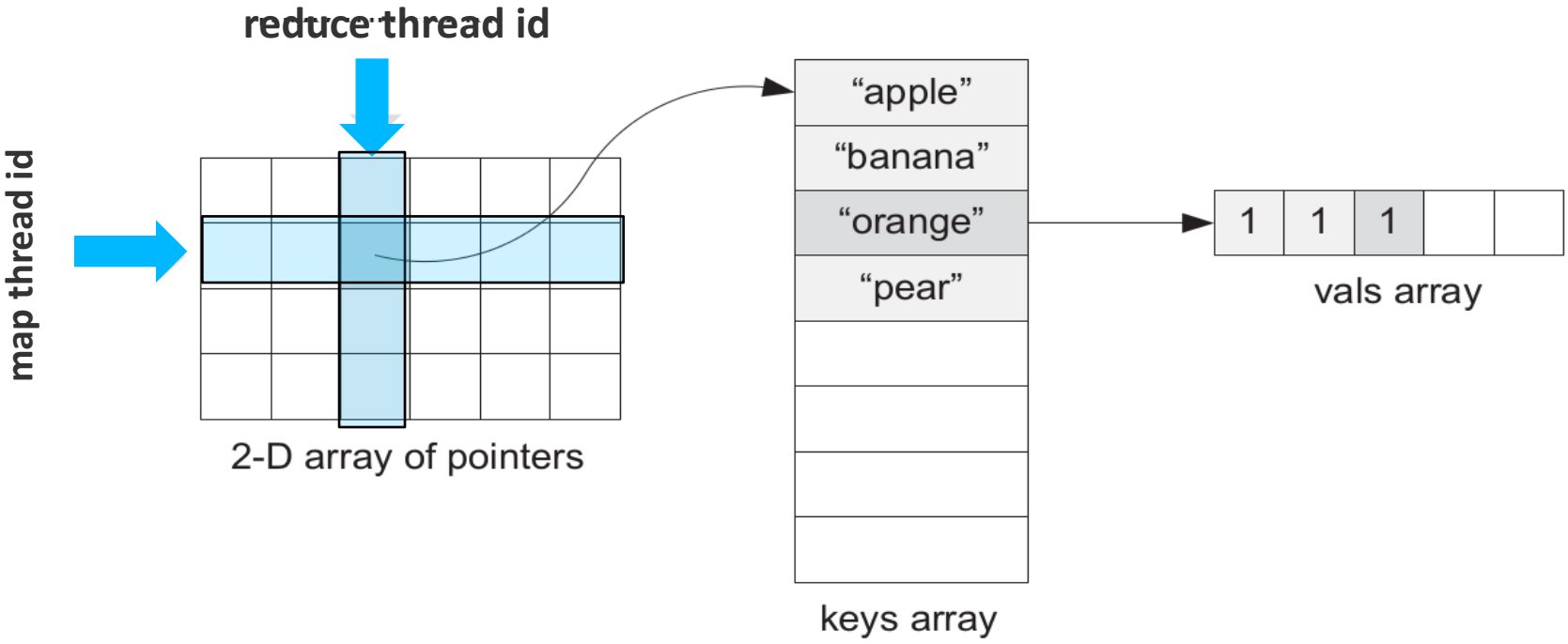
wc—wordcount

pca-pca

Phoenix存在的问题——较差的scalability

研究背景——Phoenix局限性分析之barrier

研究结果显示：多核环境下的MapReduce库，影响性能的一个关键因素是对中间结构的操作效率。

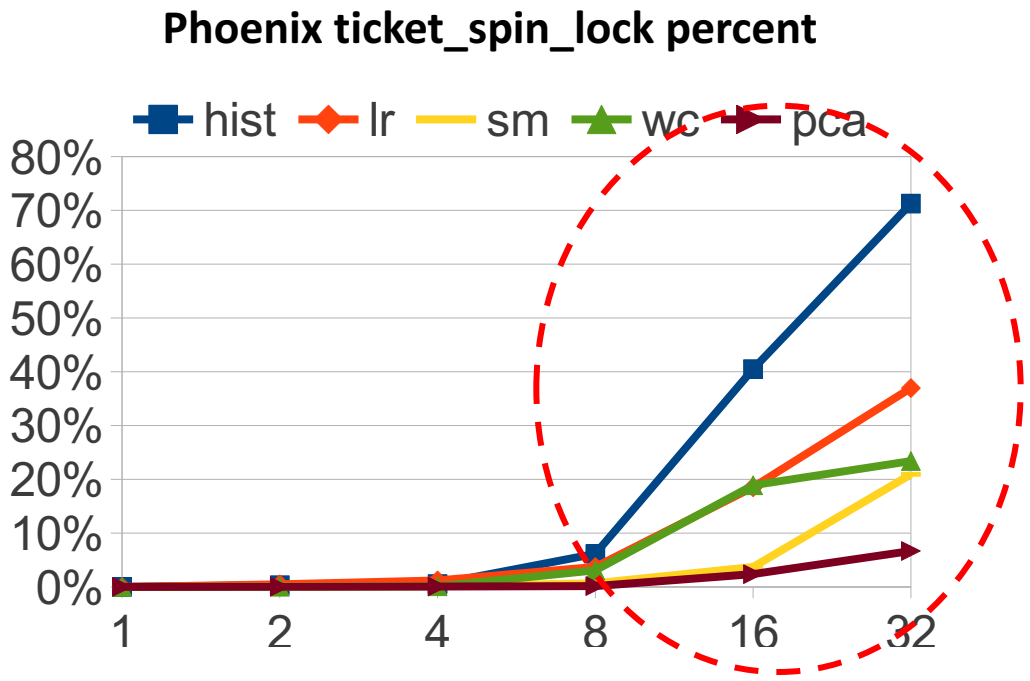


- 对全局的二维数组进行划分
- Map 和 Reduce 阶段之间加入 barrier

避免多个线程对共享区的竞争，但不利于并发执行和资源的利用

研究背景——Phoenix局限性分析之scalability

Linux Perf 测出的ticket_spin_lock的占用总运行时间的比例：



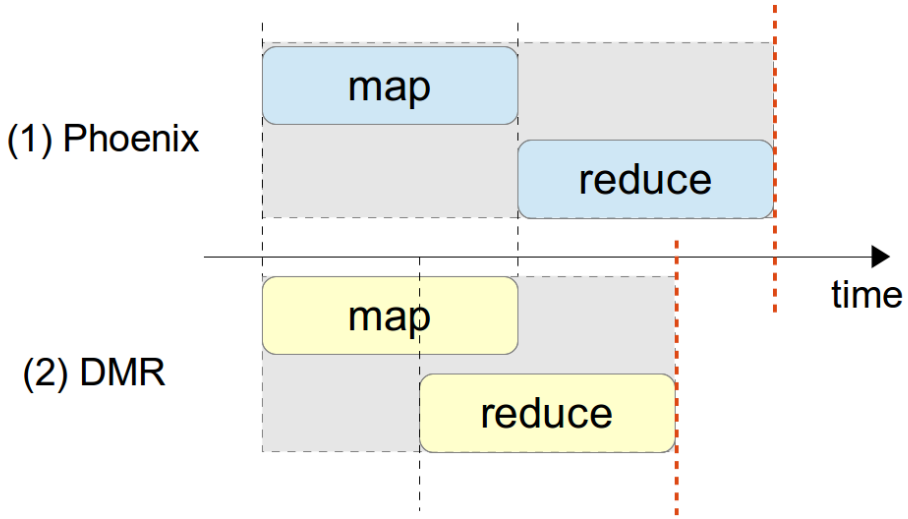
多线程共享mm_struct

高核下，Phoenix大部分的时间用于等待，而未做实际的工作，这是对多核资源的浪费

可伸缩MapReduce库SMR——设计目标

总体的目标：

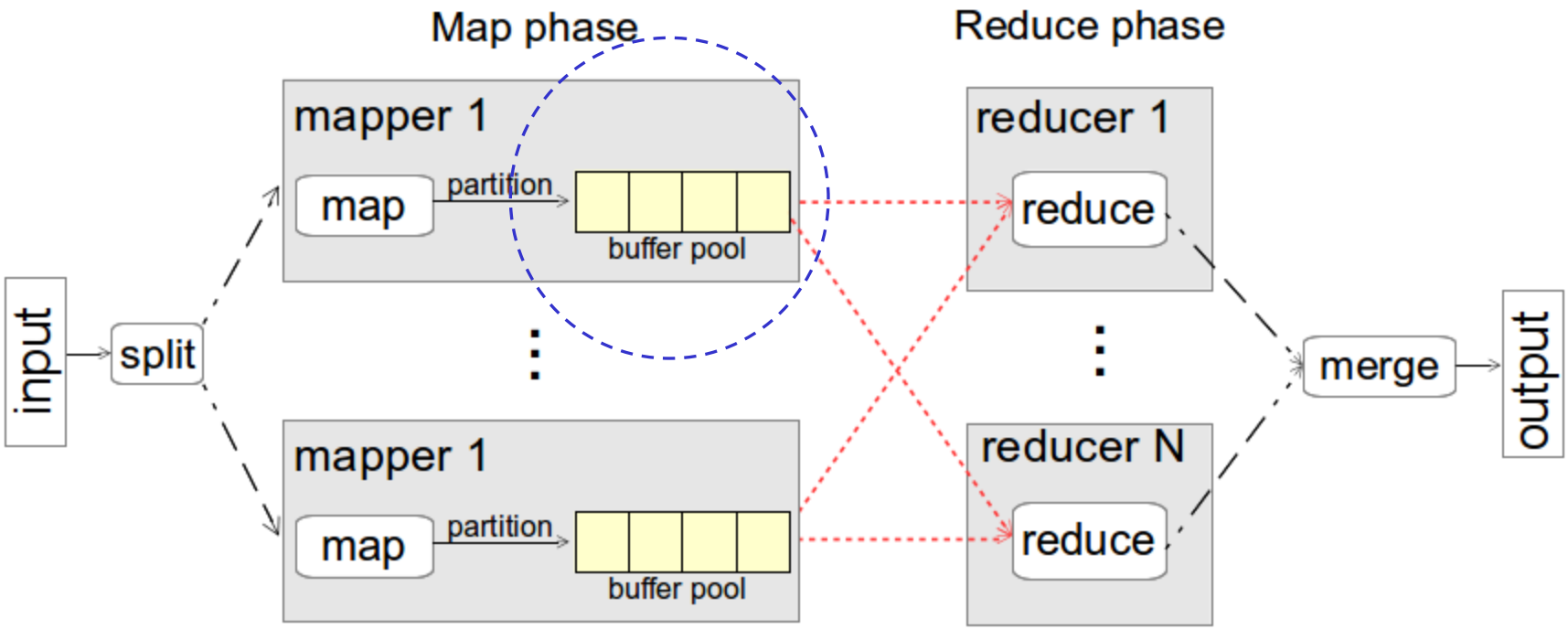
1.打破Map和Reduce阶段间的barrier，以提高性能



2.提升scalability

可伸缩MapReduce库SMR——执行流程

面向多核可伸缩的MapReduce库——SMR(Scalable MapReduce)



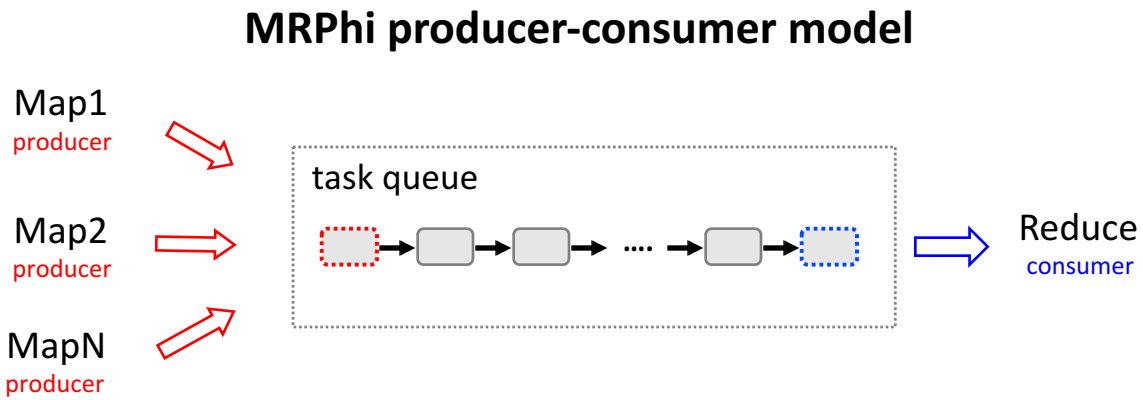
采用producer-consumer模型：

map 为producer, reduce为consumer, 私有buffer

让map worker和reduce worker拥有独立的mm_struct结构，避免竞争

SMR总体设计——producer-consumer模型

已有的MapReduce库MRPhi对producer-consumer的使用



特点和不足总结：

1.竞争问题：多个map worker需要竞争queue的尾部

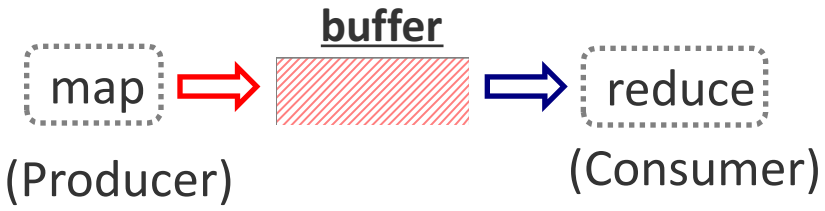
2.队列的管理问题：

固定分配：队满，map需要等待；

动态分配：会有大量动态内存分配和回收的开销

SMR总体设计——producer-consumer模型

map worker和reduce worker之间采用一对一的buffer

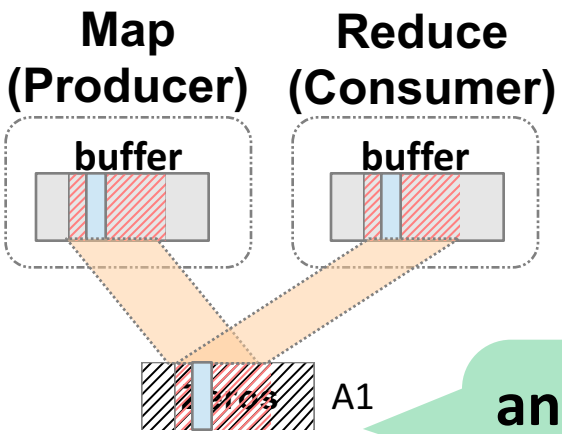


避免多个map worker的竞争

这如何做到并发执行呢？ **低层的映射机制**

SMR总体设计——低层映射机制

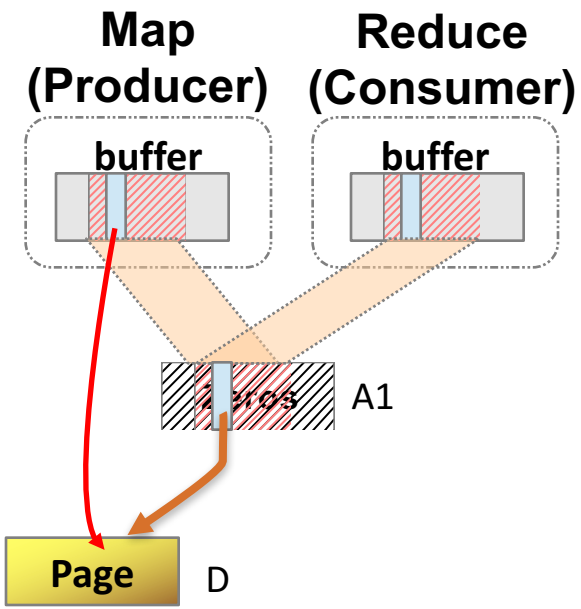
低层的映射机制



anchor的目的是为了延迟物理内存的分配

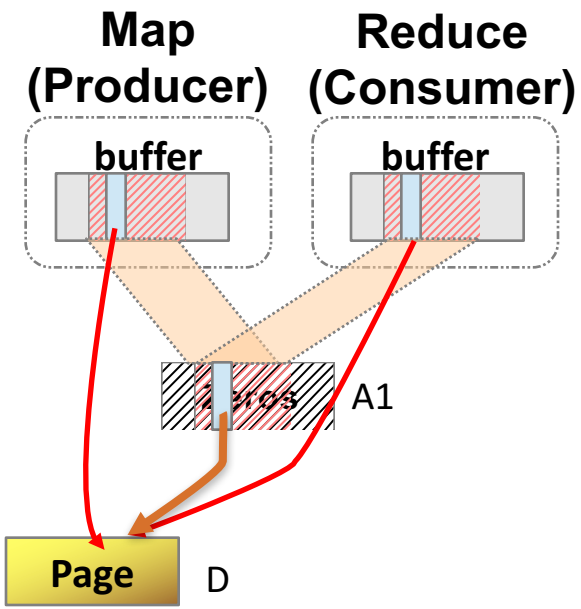
SMR总体设计——低层映射机制

低层的映射机制



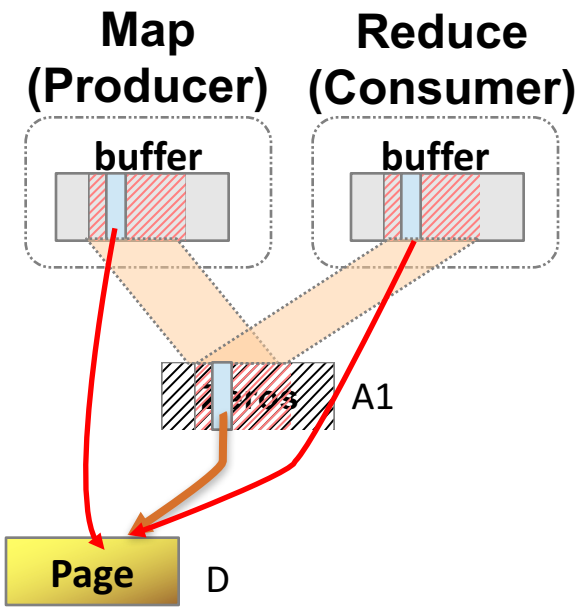
SMR总体设计——低层映射机制

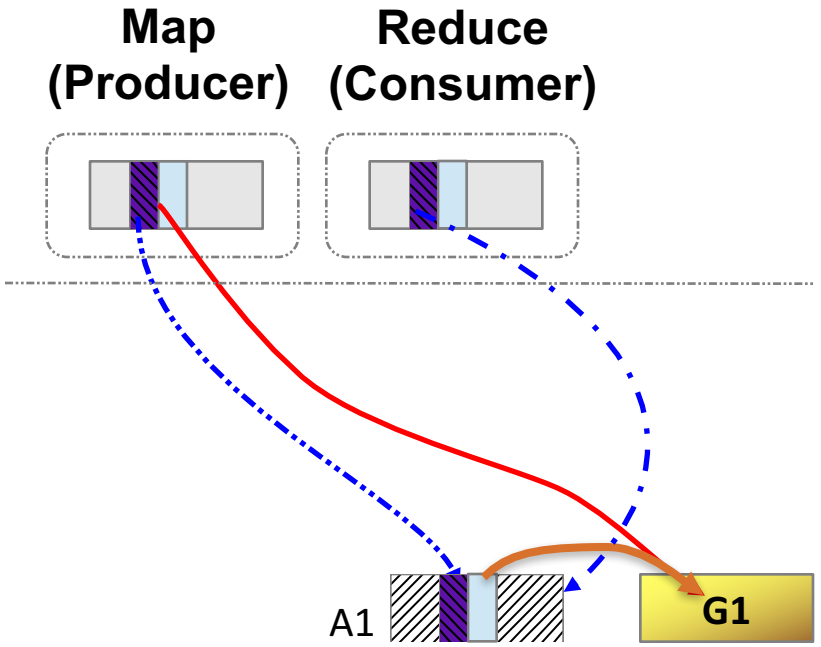
低层的映射机制



SMR总体设计——低层映射机制

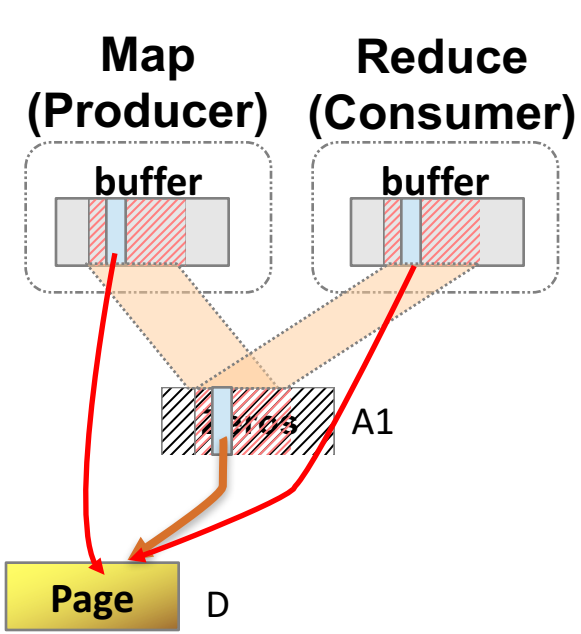
低层的映射机制

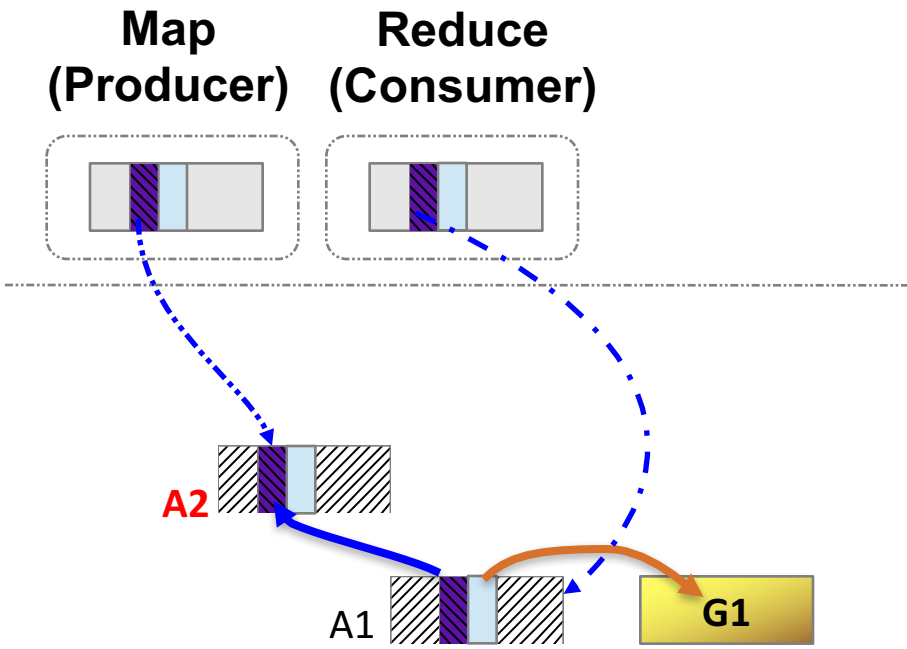




SMR总体设计——低层映射机制

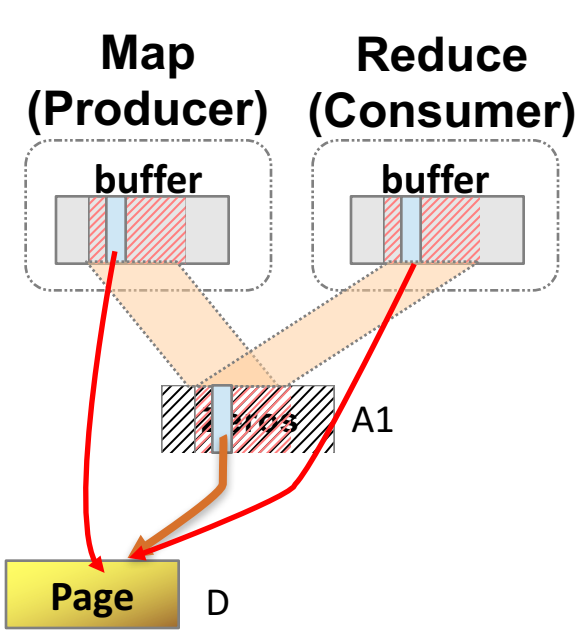
低层的映射机制

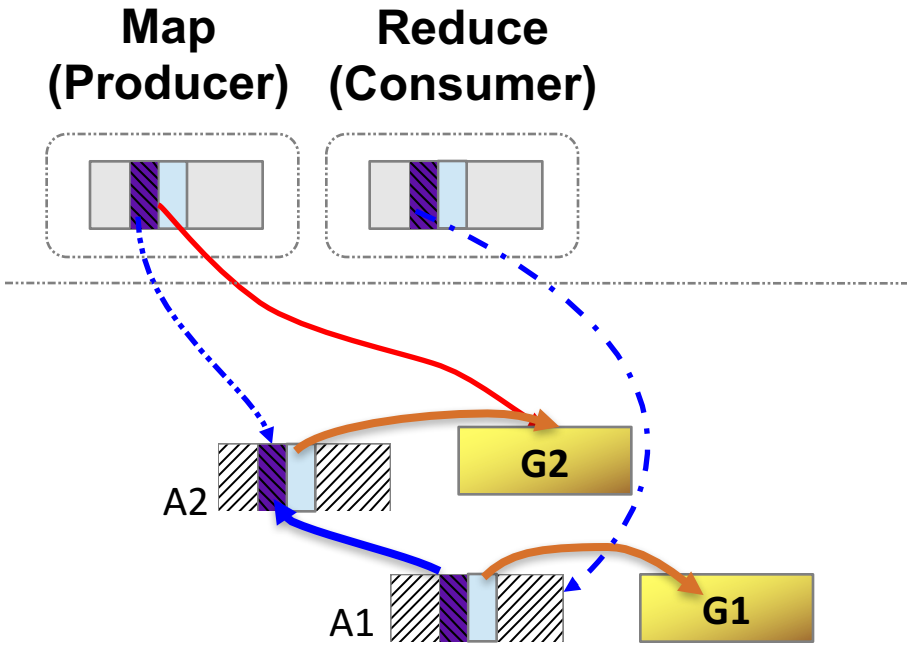




SMR总体设计——低层映射机制

低层的映射机制

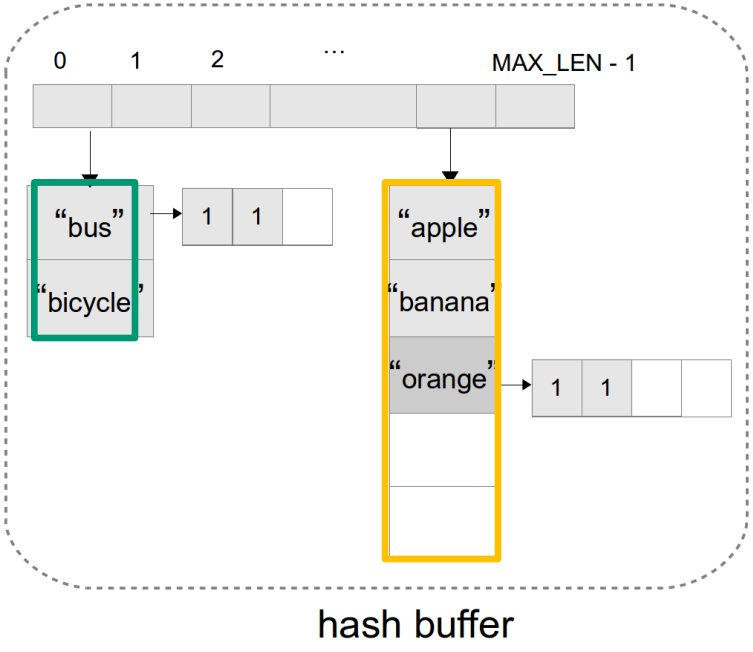




buffer满时，map worker无须等待

SMR总体设计——buffer的设计与优化

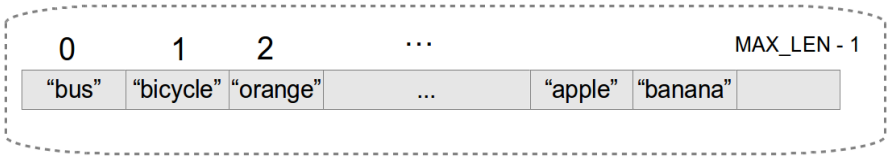
buffer的内部实现：



hash buffer

发送之前，需要将其中key-value 汇集到一块连续的地址空间

查找快速，但汇集的开销大

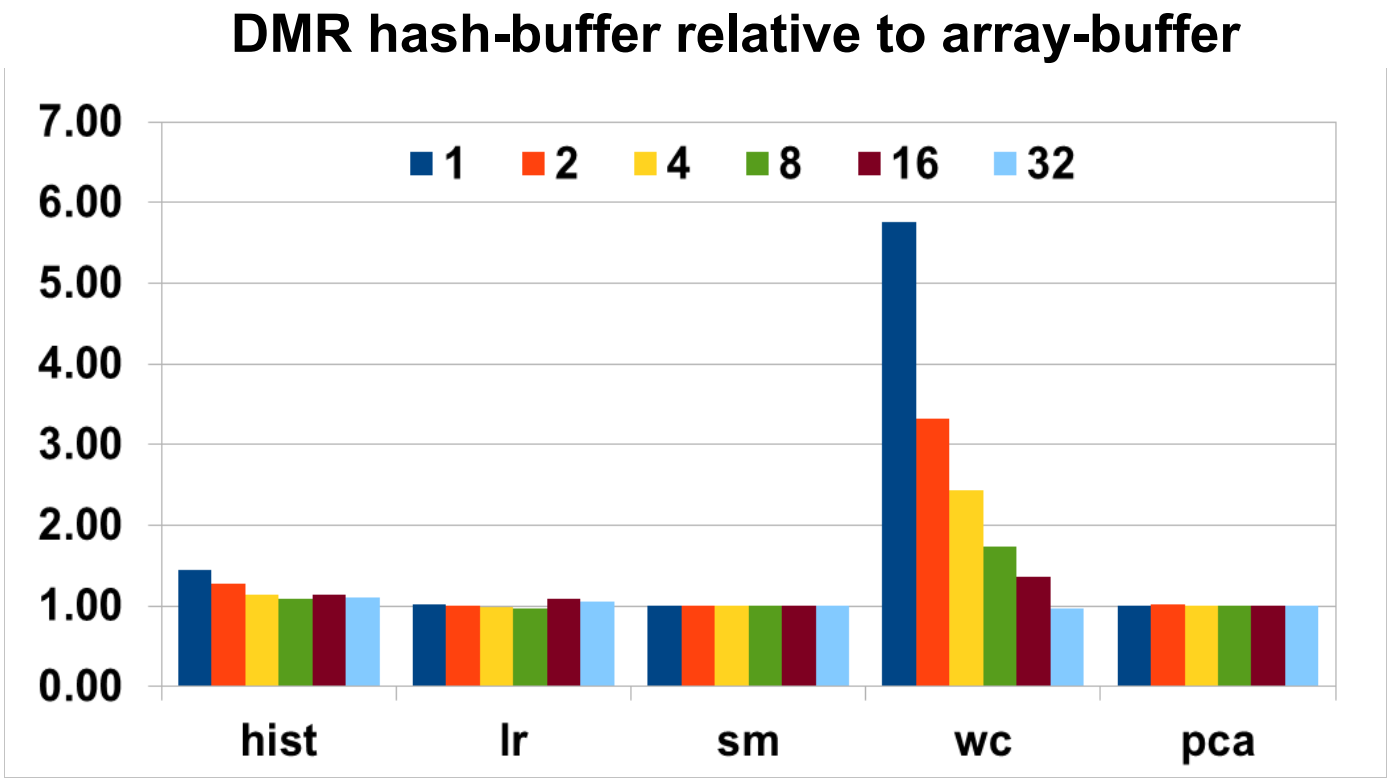


array buffer

查找慢，但无须汇集

SMR总体设计——私有buffer实现的优化

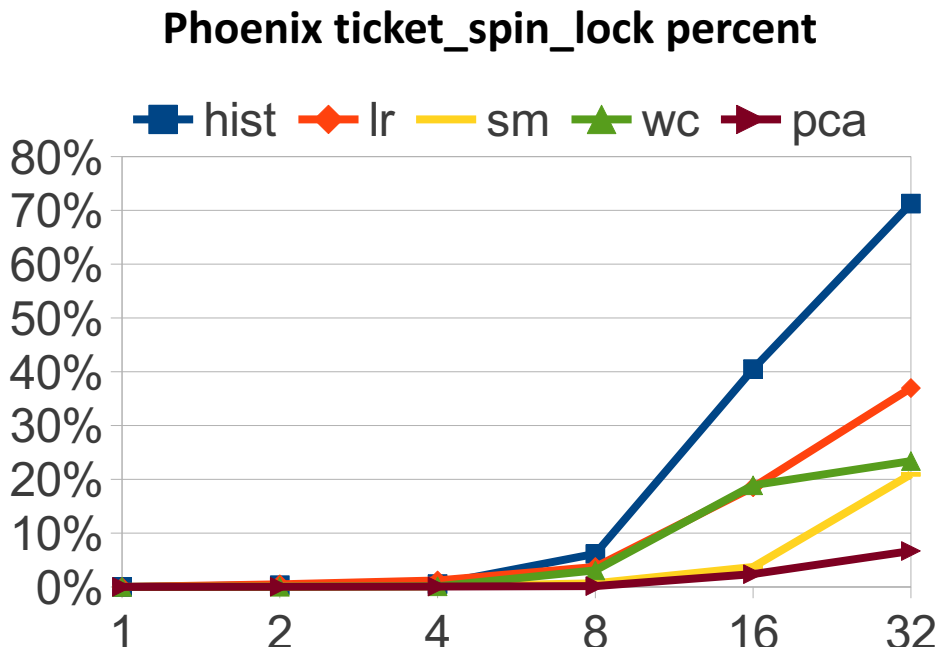
不同buffer的对比结果：



wc使用array-buffer性能更好，wc中存在大量的key-value导致汇集的开销较大

研究背景——Phoenix局限性分析之scalability

Linux Perf 测出的ticket_spin_lock的占用总运行时间的比例：

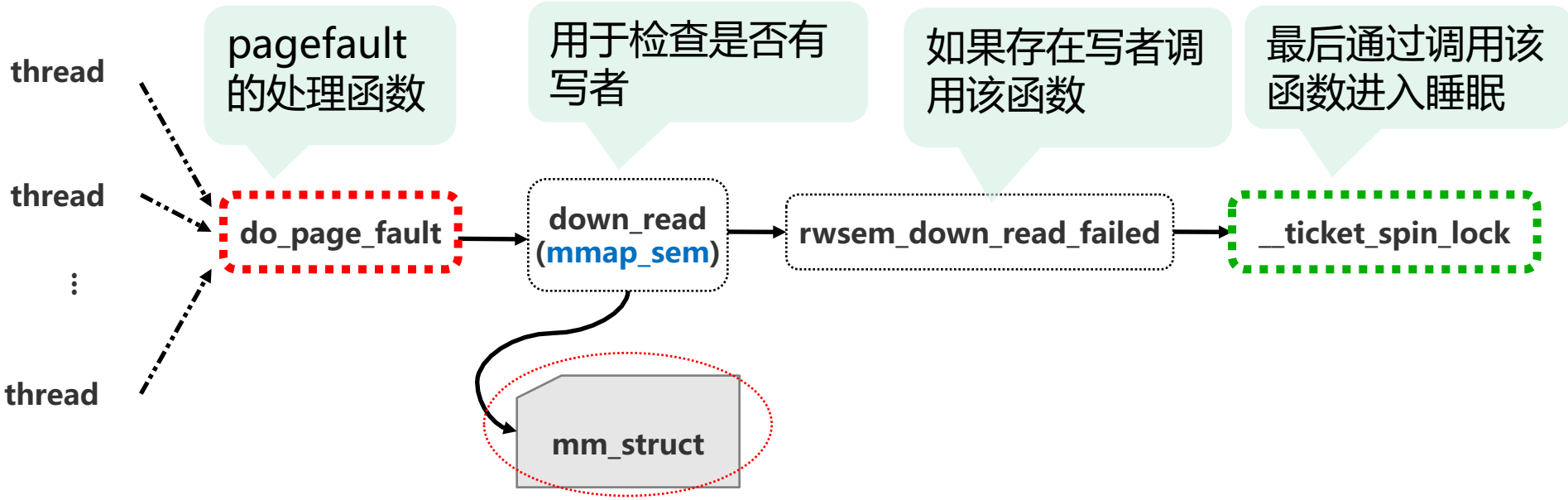


多线程共享mm_struct

高核下，Phoenix大部分的时间用于等待，而未做实际的工作，这是对多核资源的浪费

SMR总体设计——Scalability的优化

Linux Perf record记录的函数调用栈:

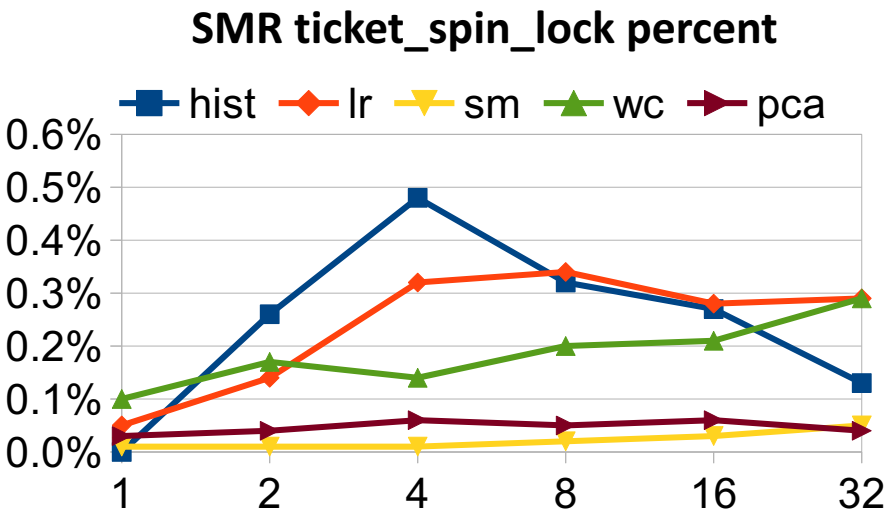


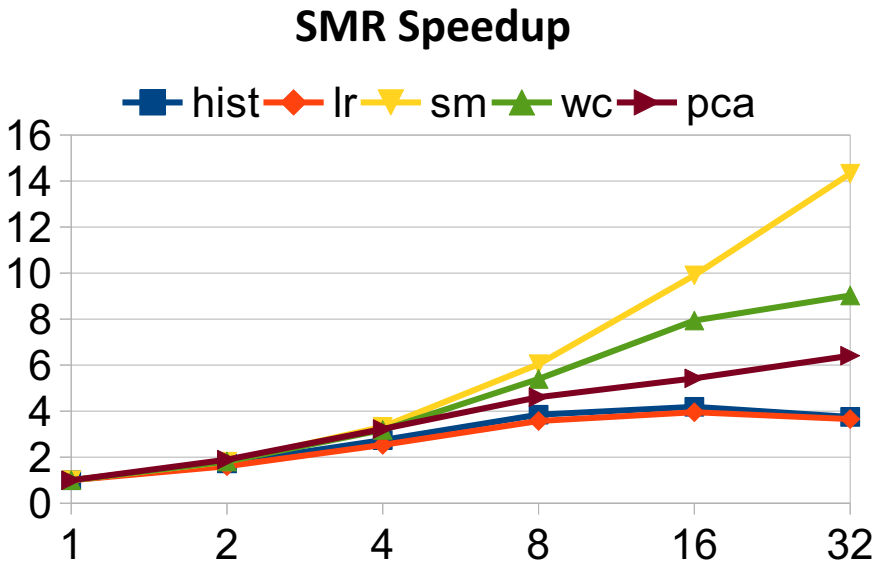
SMR中不再使用线程实现，而是采用进程，进程地址空间隔离，多个进程之间不需要竞争mmap_sem信号量

采用进程实现，会有什么问题呢？**不便于数据的共享？开销大？**

实验结果分析——SMR的scalability

基于SMR的应用程序的性能



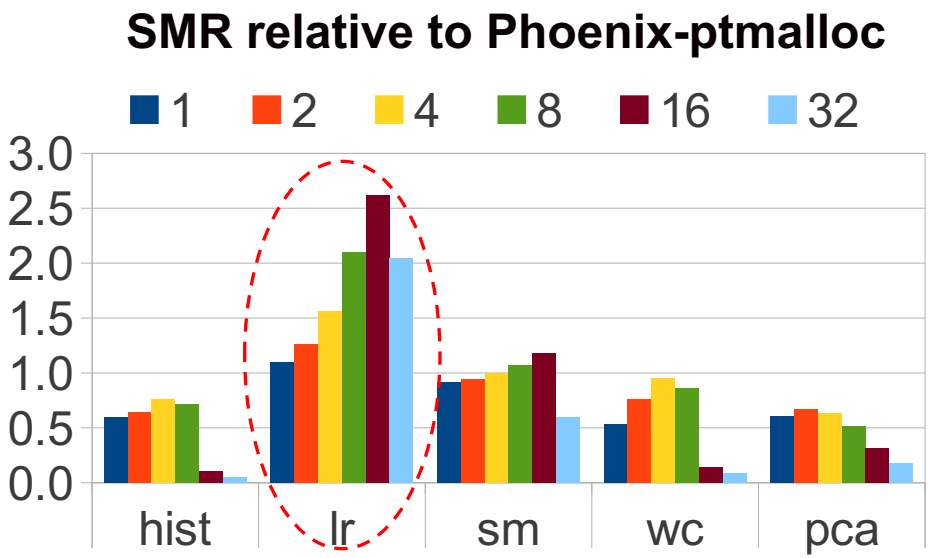


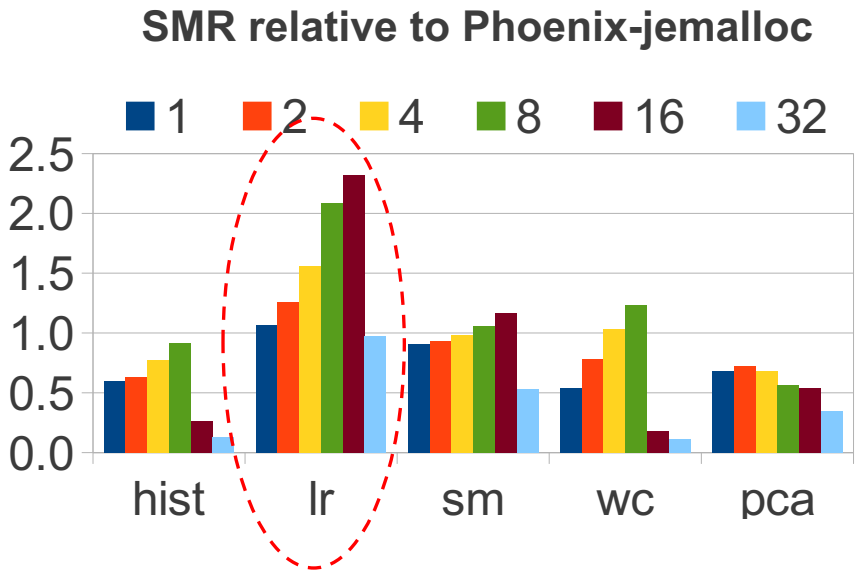
ticket_spin_lock占用比例相当低
随着核数的增多，没有出现显著的上升

Sm,wc,pca的scalability相当好
lr和hist的16核以后，性能提升有限

实验结果分析——总体性能

实验结果(环境32-core 2.0GHz 4*Xeon E7-4820, ubuntu12.04, gcc 4.4.7**)**



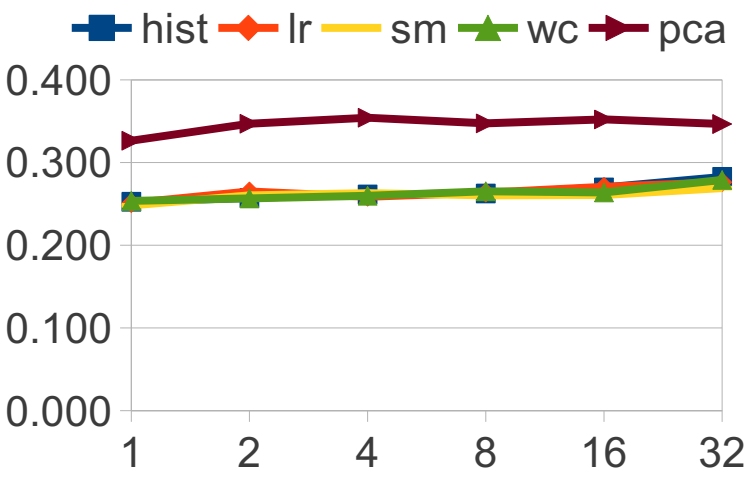


- 基于SMR的hist, wc, pca都具有较好的性能
- 基于SMR的lr的性能较差

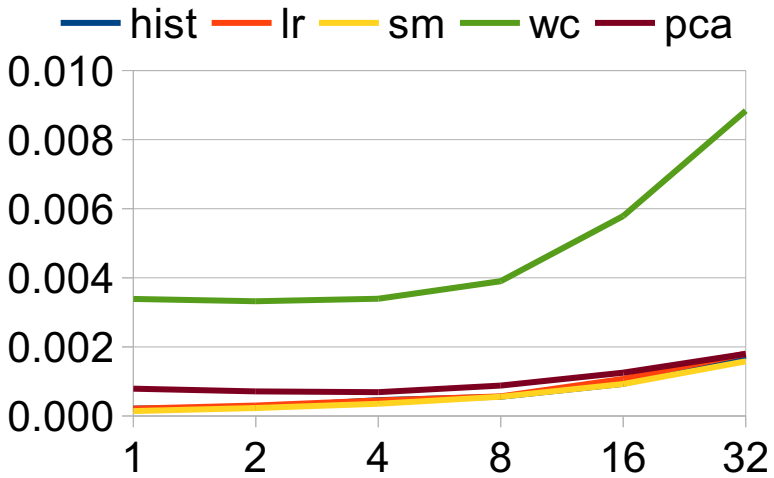
实验结果分析——SMR额外开销分析

SMR初始化的开销较大

environment initialize time(seconds)



(a) SMR

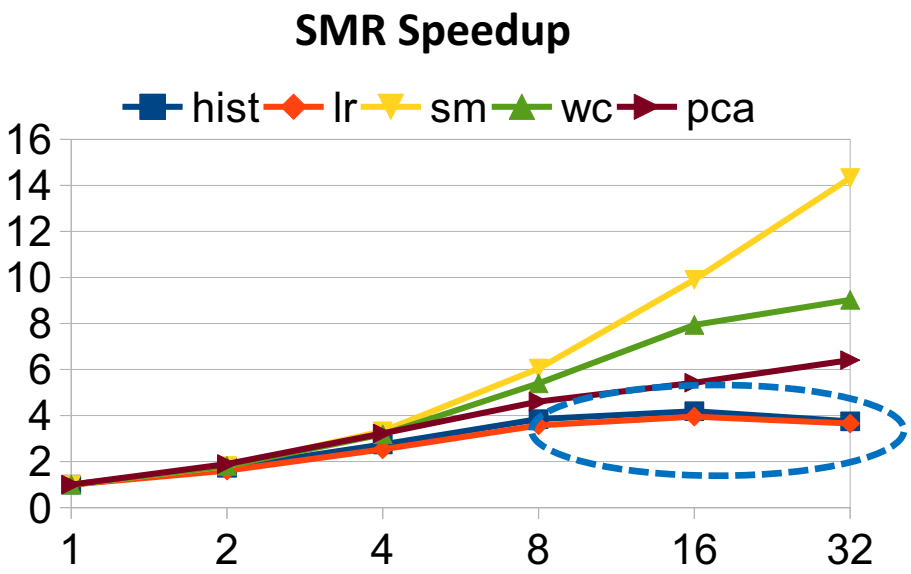


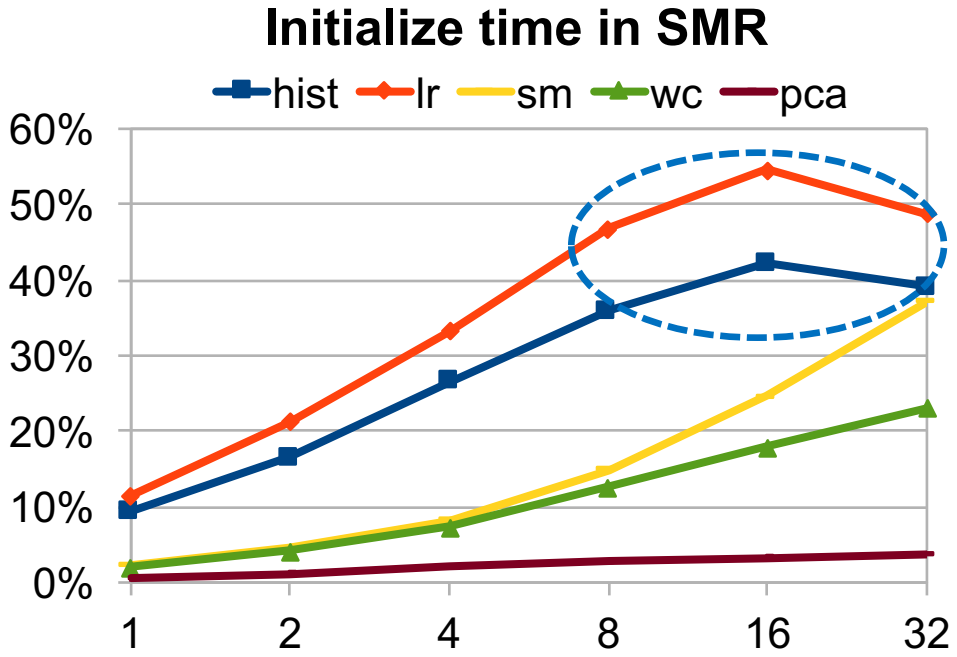
(b) Phoenix

针对数据集较大、总运行时间较长的应用程序，DMR的性能优势就越是明显

实验结果分析——SMR额外开销的分析

基于SMR的性能和scalability的分析





lr和hist性能上升不明显的原因是，初始化的开销占用总时间的比例较大

总结

我们提出了一个具有较好scalability的MapReduce库SMR

1.Producer-consumer模型，提高性能

2.使用进程避免多个线程的mm_struct的竞争，从而具有较好的scalability

中国科学技术大学
University of Science and Technology of China

Q&A

Thanks for listening

报告后的提问（2016-11-21）

1问：pagefault真的会引起那么大的ticket_spin_lock吗，
真的会有那么多的pagefault吗？相比等待pagefault在ticket_spin_lock的等待的时间，
用在处理pagefault的时间应该更多吧？
我的思考：首先别人提出这个问题说明，我报告中的解释以及我的论据并不充足，
别人怀疑我的数据和我的分析。
首先，我必须对内核的pagefault的处理要非常熟悉，
其次，应该结合应用程序的特点，多线程的特点，linux内核中处理pagefault的整个流程，
综合解释ticket_spin_lock高，简单的说是因为竞争mm_struct结构，会让人觉得不可信。
最后，给出实验结果

2问：为什么要map和reduce并发执行，combiner明明就已经是reduce了，
为什么还要让reduce提前？真的能提升性能吗？
首先，在接下来写的论文中，不可以模糊combiner的概念，
需要指名提前reduce有什么好处，并通过详细的实验数据证明

3考虑：对spmc的生产者和消费者模型的理解不够，接下来的论文中，
考虑如何结合SMR来详细的清晰的表述这个模型的特点和亮点