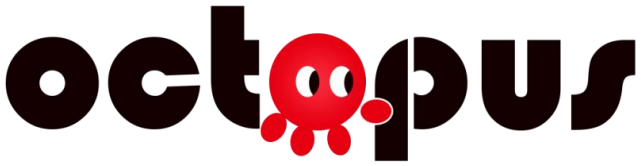
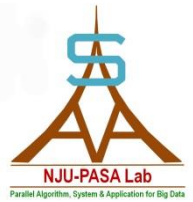


A High-level Unified Programming Model and Platform for Big Data Analytics

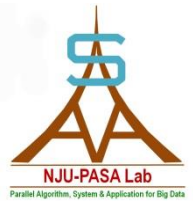
顾 荣
南京大学 PASA大数据实验室
2015/07/25@Nanjing BigData Tech Meetup

内容



- 研究动机与目标
- 大章鱼的编程模型与系统架构
- 大章鱼的系统实现与优化
- 性能评估
- Demo & 总结

内容



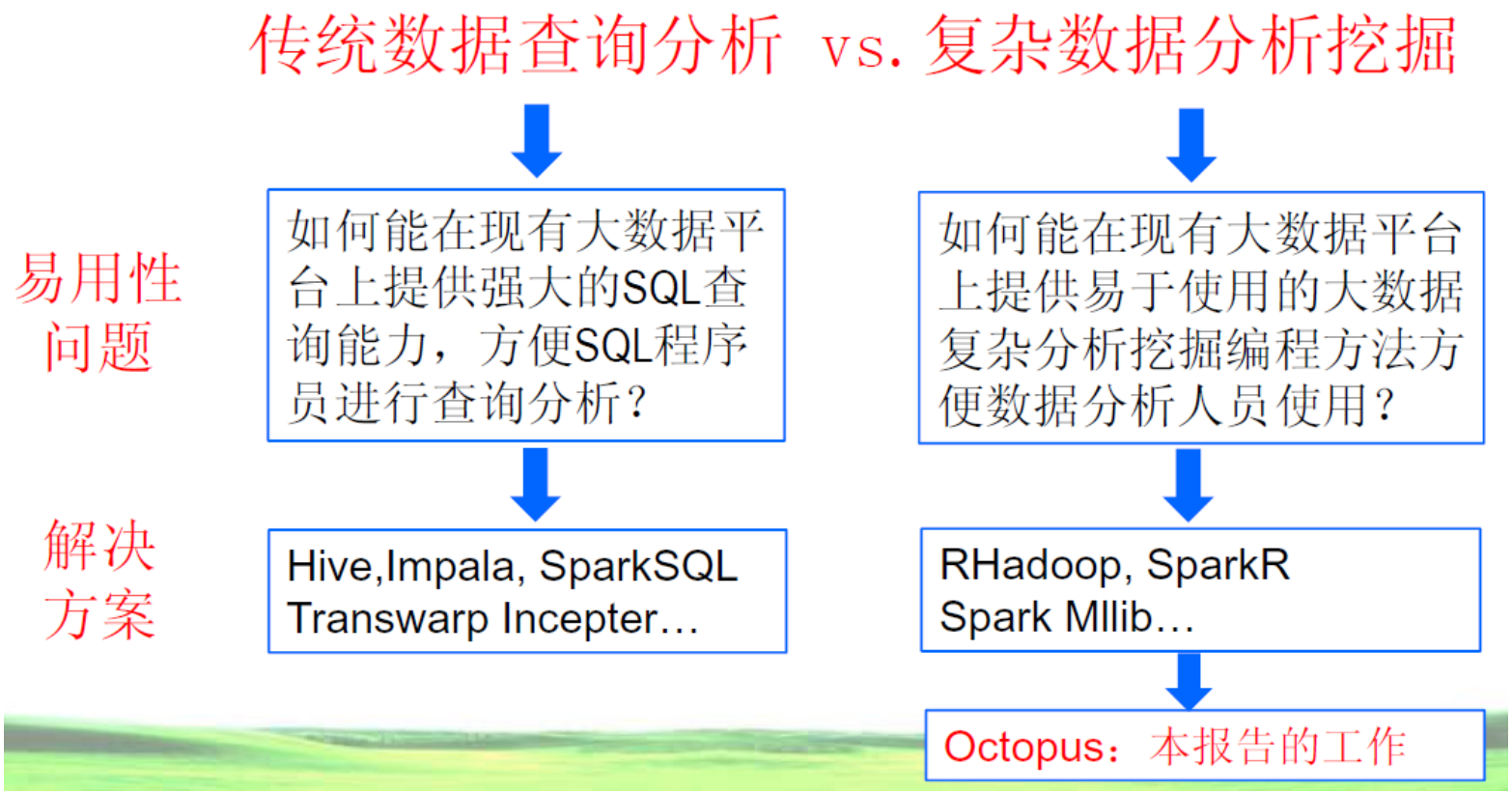
- 研究动机与目标
- 大章鱼的编程模型与系统架构
- 大章鱼的系统实现与优化
- 性能评估
- Demo & 总结

研究动机与目标

- 大数据处理平台从无到有、从慢到快，下一步要重点解决的是从难于使用到易于使用

研究动机和目标

- 大数据处理的两大计算类型



研究动机和目标

NJU-PASA Lab
Parallel Algorithm, System & Application for Big Data

基于矩阵/向量的建模

$$\begin{pmatrix} a_{11} \\ a_{12} \cdots \\ a_{21} \\ a_{22} \cdots \\ \cdots \end{pmatrix}$$

[a1, a2, a3, a4, ...]

大数据分析人员

擅长使用的分析工具

MATLAB
The Language of Technical Computing

R

A Big Gap !

大数据处理平台与并行程序设计

MPI、Fortran/C++ ScaLAPACK ;
GPU CUDA、BIDMach ;
Scala、Spark RDD;
Hadoop MR;

Spark

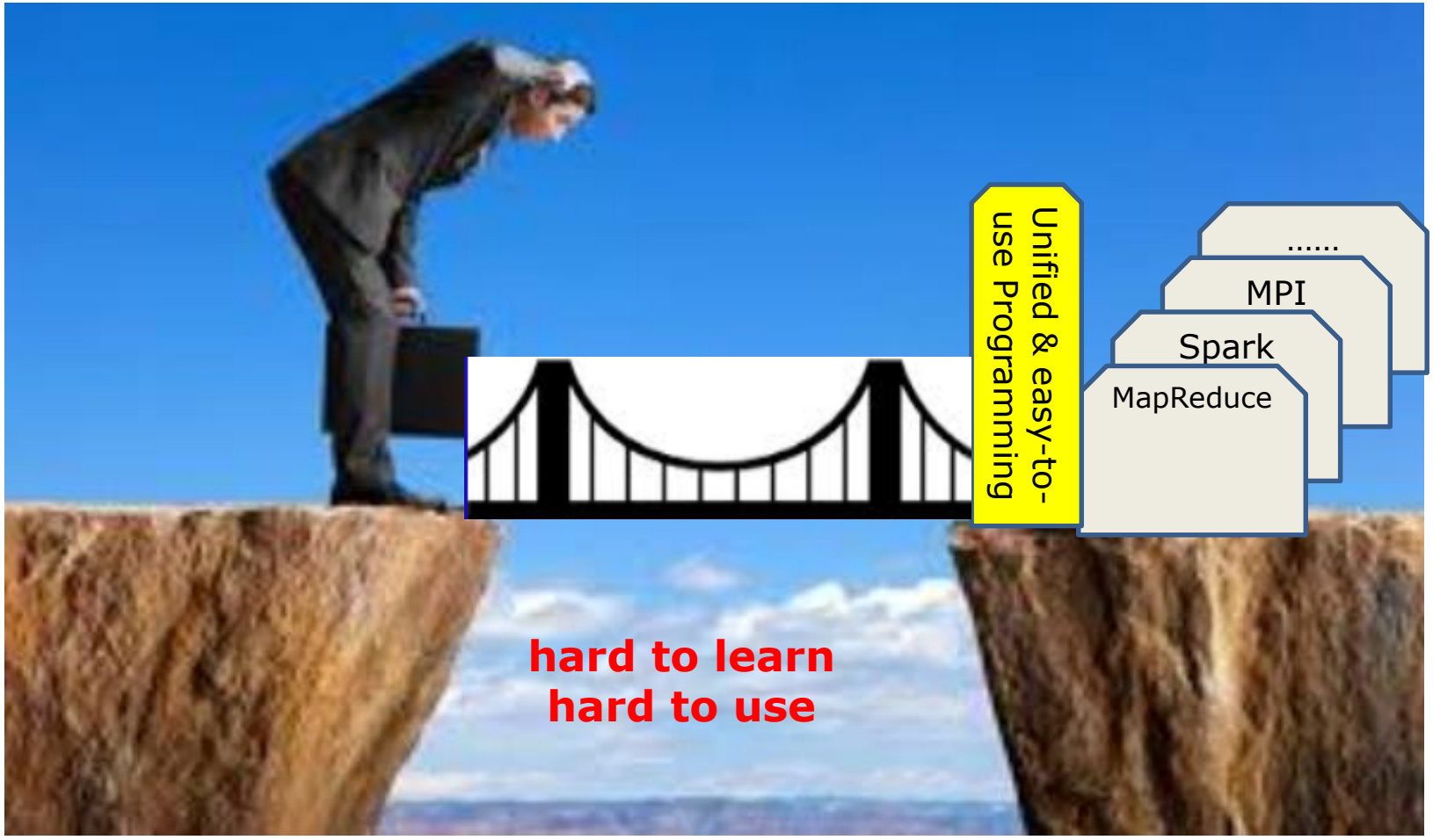
OPEN MPI

hadoop

Flink

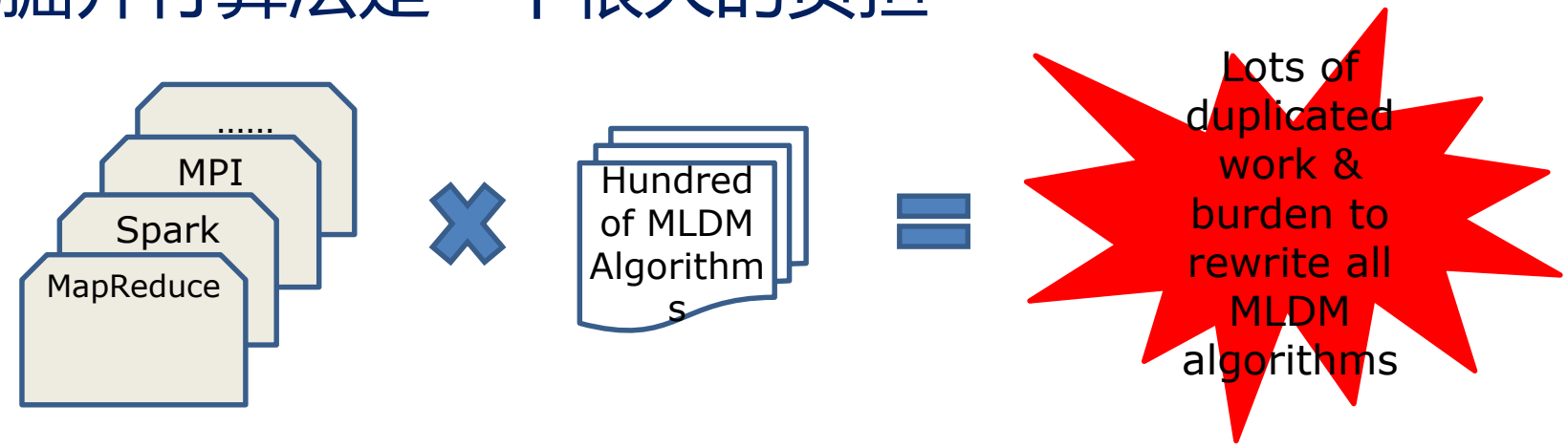
研究动机和目标

- 对于应用行业的数据分析师来说，现有的大数据处理平台不易学习和掌握使用



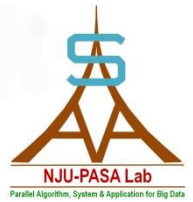
研究动机和目标

即使对于专业的大数据应用开发程序员来说，在多个大数据平台上重复性设计实现大量的机器学习和数据挖掘并行算法是一个很大的负担



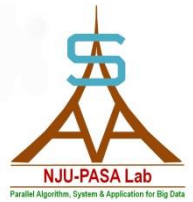
通过提供跨平台统一的大数据机器学习和数据挖掘编程模型和框架，避免重复编程，实现“Write Once，Run Anywhere”

内容



- 研究动机与目标
- 大章鱼的编程模型与系统架构
- 大章鱼的系统实现与优化
- 大章鱼的性能评估
- Demo & 总结

编程模型的基本设计思想

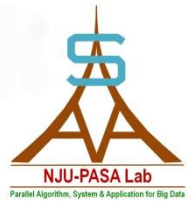


- 很多机器学习与数据挖掘算法都可以基于矩阵/向量建模；
  - 采用矩阵作为各种算法的抽象计算和编程模型
  - 提供一个基于矩阵的高层并行化编程和计算模型
- 为数据分析师和专业开发程序员提供一个基于R语言和矩阵模型的跨平台统一编程模型和框架

编程模型

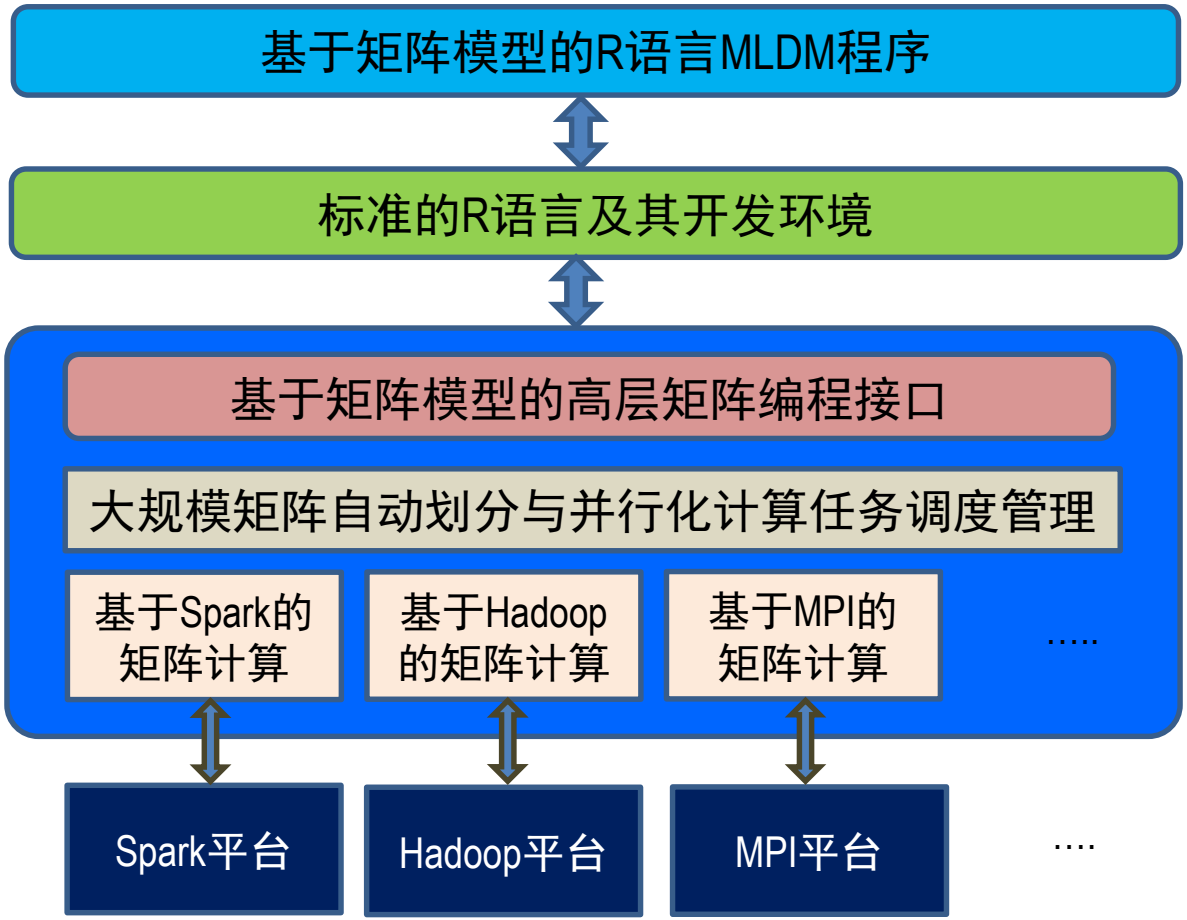
- 提供一个常规的矩阵数据模型：OctMatrix
- 支持多种常用的矩阵操作/计算的接口：
  - 矩阵的初始化、保存等操作；
    - 从外部数据源读取、内存构建
    - ones, zeros
  - BLAS 1-3 级别的向量、矩阵的运算
    - 向量/向量、向量/矩阵、矩阵/矩阵的相加、相乘等
  - 更高级线性代数运算
    - 求矩阵特征值，矩阵各种分解，求逆、求解线性方程组等
  - R特有的一些便捷的矩阵的操作
    - apply, split, rep, cbind2,min,max,mean,sum等
  - 机器学习常用的对矩阵数据的操作
    - SVD分解、按行求SGD等；

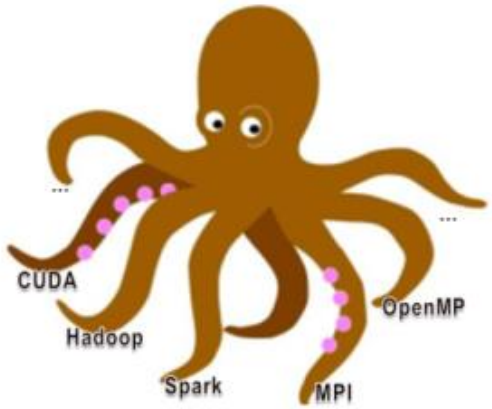
编程模型的基本设计思想



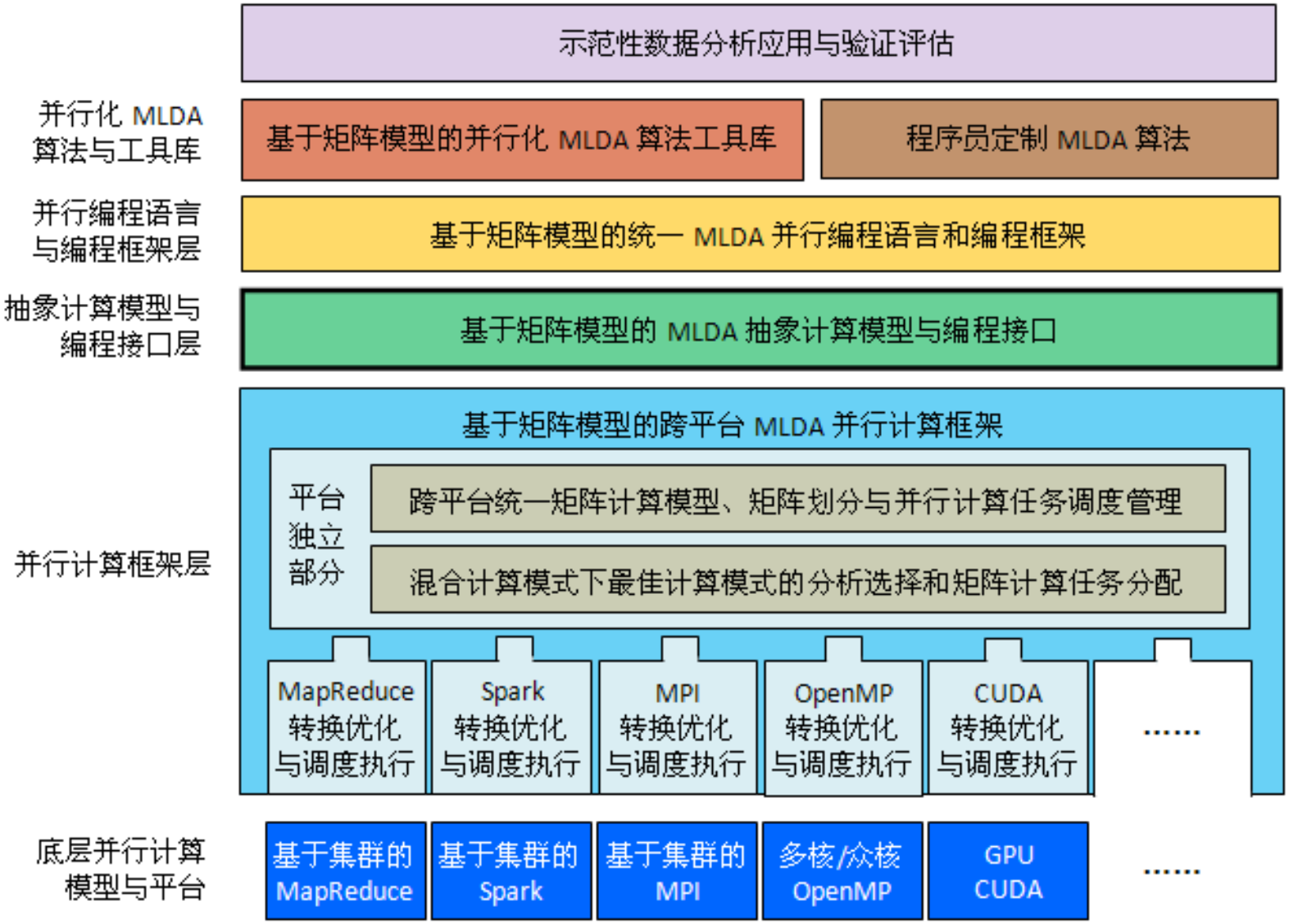
- 为底层各种异构大数据处理平台实现优化的大规模矩阵分布和并行计算和任务调度管理
  - 实现底层异构平台对程序员的透明化；
  - 实现“Write once, run anyway”的目标；
- 在此基础上设计提供基于矩阵模型和统一平台的并行化机器学习算法库

大章鱼系统执行框架示意图

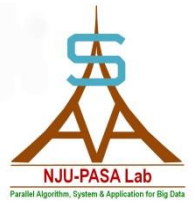




概念性总体框架示意

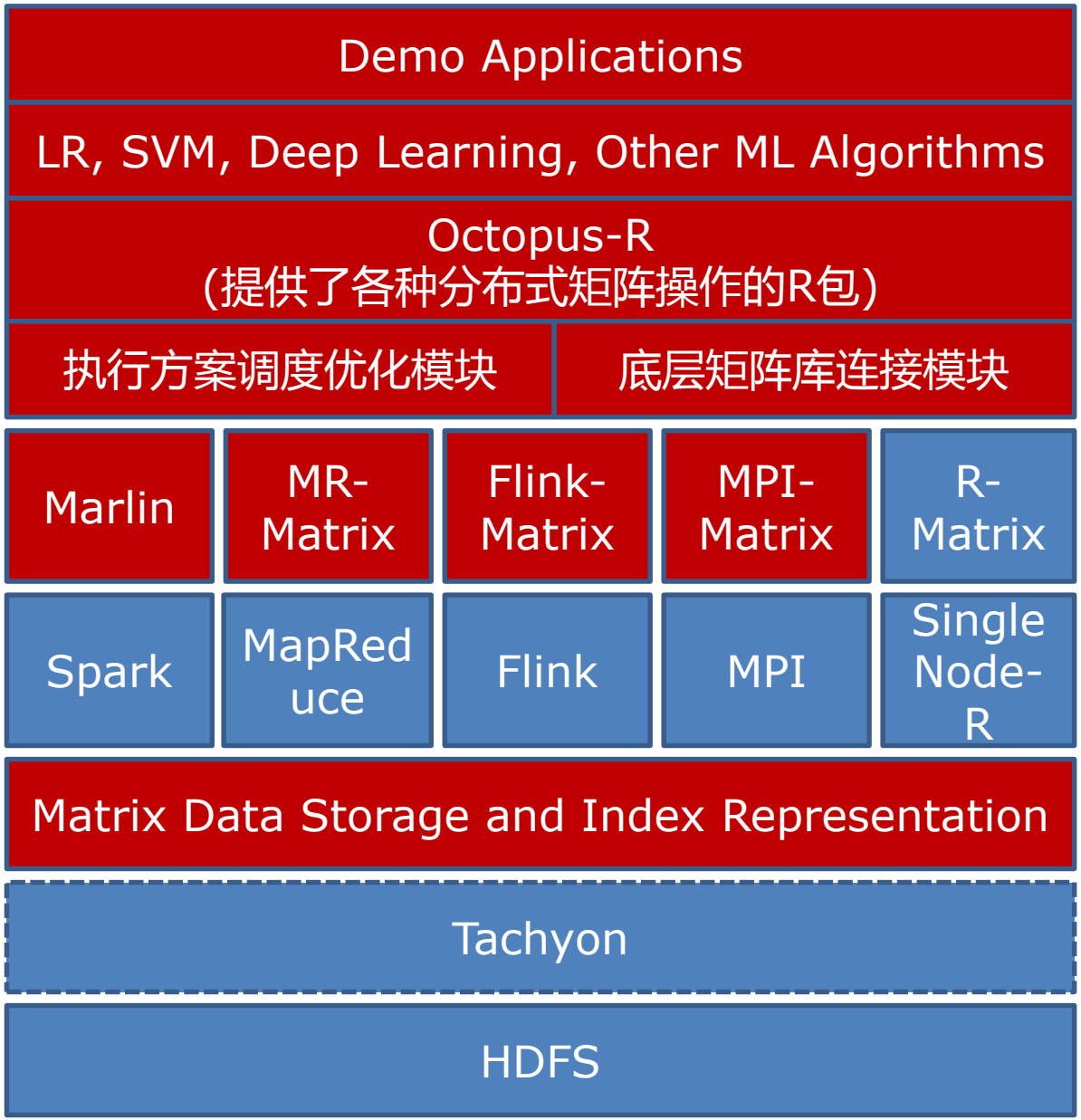

示范性数据分析应用与验证评估
并行化 MLDA 算法与工具库
基于矩阵模型的并行化 MLDA 算法工具库
程序员定制 MLDA 算法
并行编程语言与编程框架层
基于矩阵模型的统一 MLDA 并行编程语言和编程框架
抽象计算模型与编程接口层
基于矩阵模型的 MLDA 抽象计算模型与编程接口
并行计算框架层
基于矩阵模型的跨平台 MLDA 并行计算框架
平台独立部分
跨平台统一矩阵计算模型、矩阵划分与并行计算任务调度管理
混合计算模式下最佳计算模式的分析选择和矩阵计算任务分配
MapReduce 转换优化与调度执行
Spark 转换优化与调度执行
MPI 转换优化与调度执行
OpenMP 转换优化与调度执行
CUDA 转换优化与调度执行
.....
底层并行计算模型与平台
基于集群的 MapReduce
基于集群的 Spark
基于集群的 MPI
多核/众核 OpenMP
GPU CUDA
.....

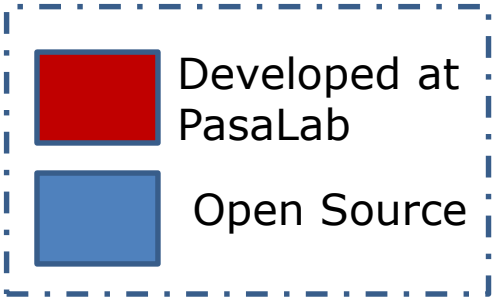
内容



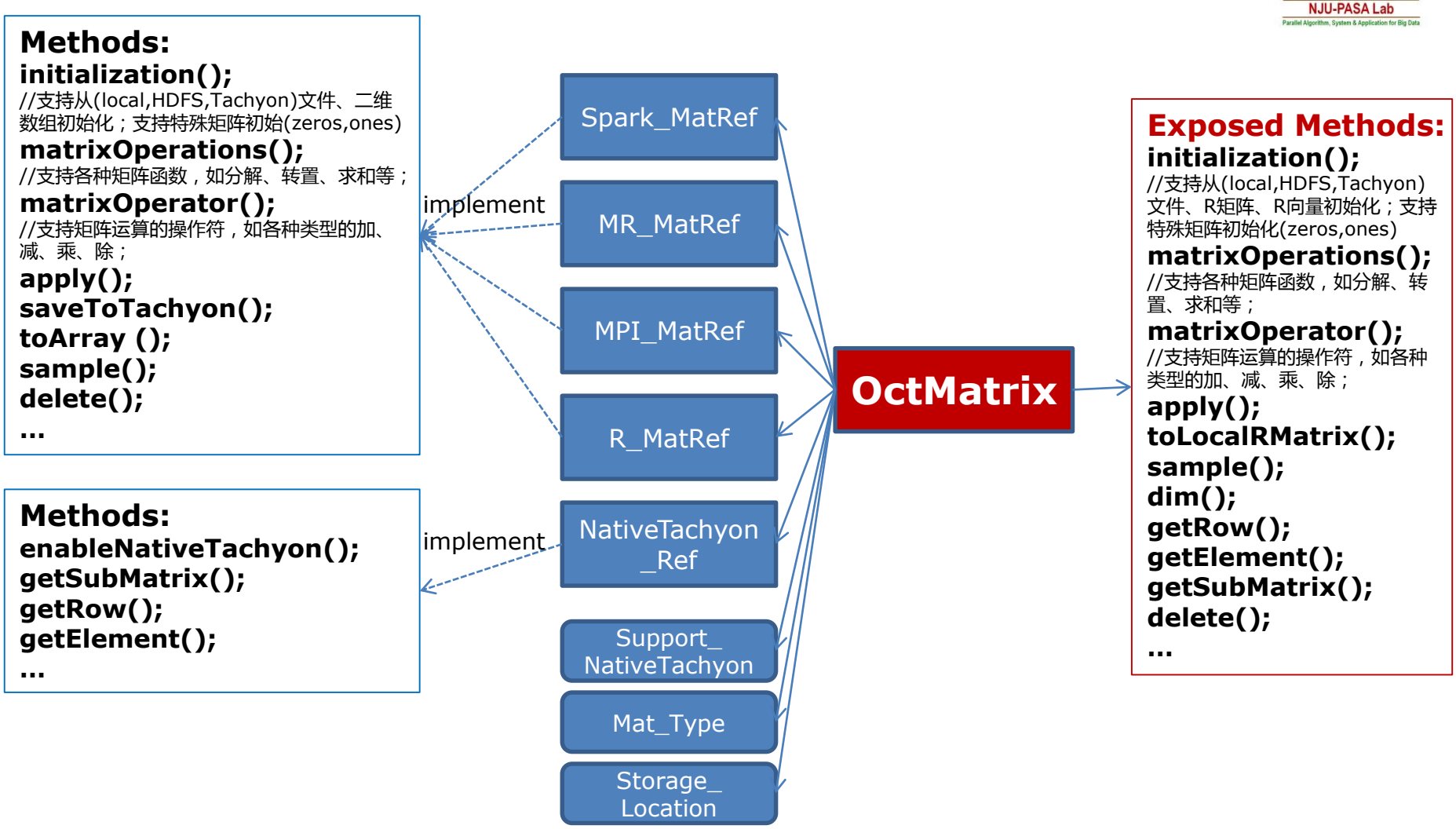
- 研究动机与目标
- 大章鱼的编程模型与系统架构
- 大章鱼的系统实现与优化
- 大章鱼的性能评估
- Demo & 总结

大章鱼系统实现模块图


Demo Applications
LR, SVM, Deep Learning, Other ML Algorithms
Octopus-R
(提供了各种分布式矩阵操作的R包)
执行方案调度优化模块
底层矩阵库连接模块
Marlin
MR-Matrix
Flink-Matrix
MPI-Matrix
R-Matrix
Spark
MapReduce
Flink
MPI
Single Node-R
Matrix Data Storage and Index Representation
Tachyon
HDFS

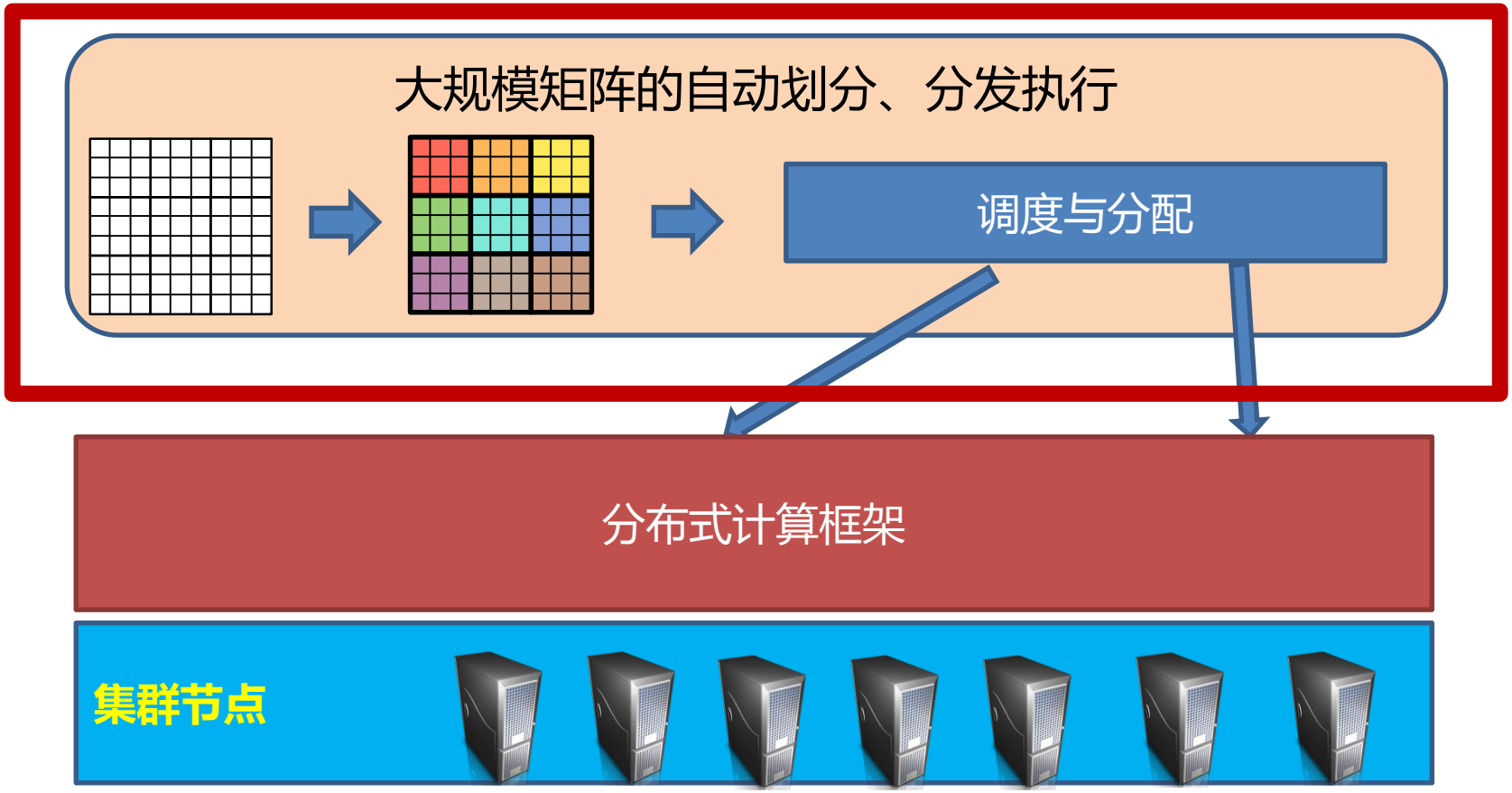

Developed at PasaLab
Open Source

Octopus-R包的实现架构

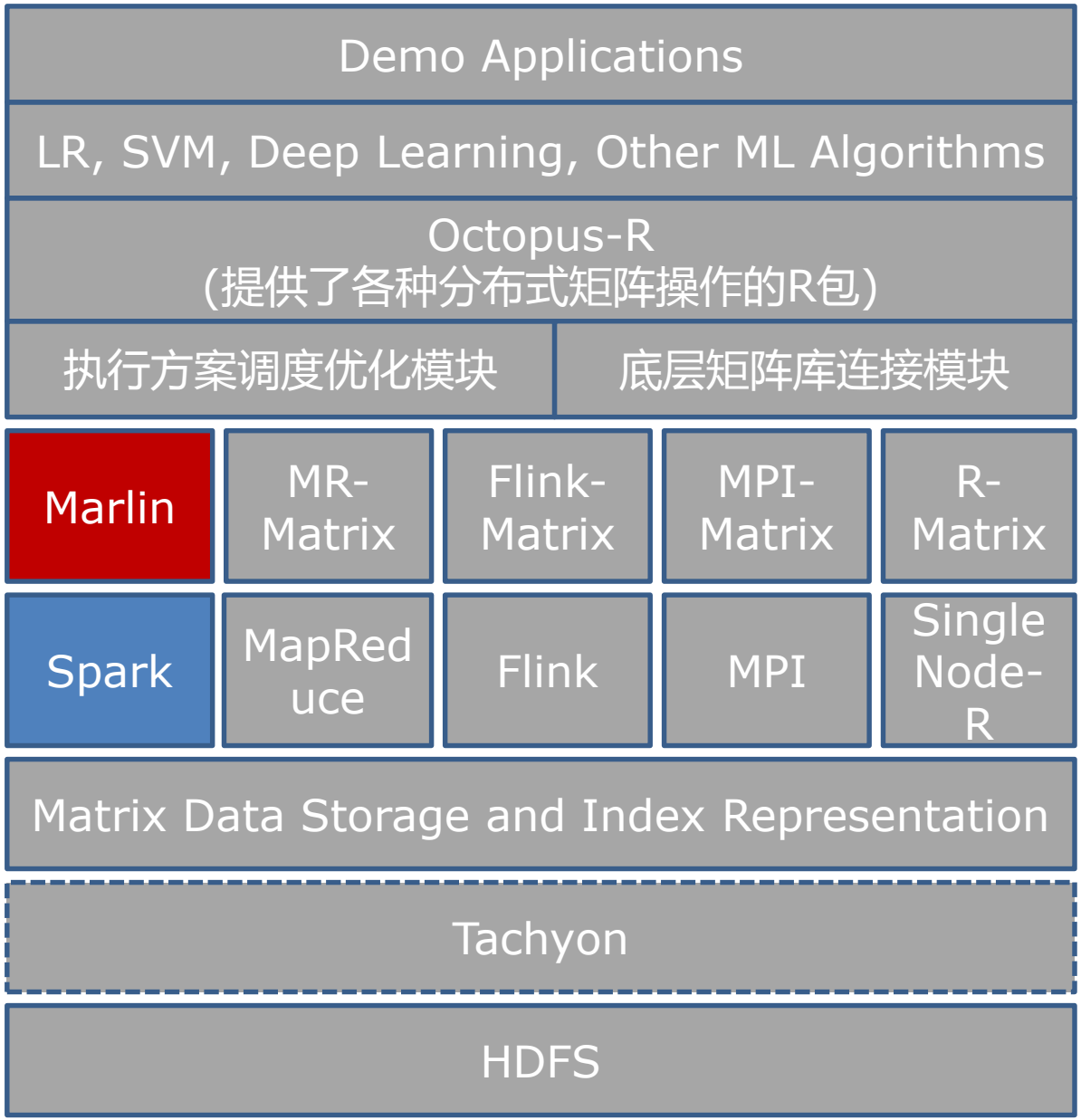


矩阵的分布式计算

- 基本手段是分而治之
  - Blocking策略



NJU-PASA Lab
Parallel Algorithm, System & Application for Big Data


Demo Applications
LR, SVM, Deep Learning, Other ML Algorithms
Octopus-R
(提供了各种分布式矩阵操作的R包)
执行方案调度优化模块
底层矩阵库连接模块
Marlin
MR-Matrix
Flink-Matrix
MPI-Matrix
R-Matrix
Spark
MapReduce
Flink
MPI
Single Node-R
Matrix Data Storage and Index Representation
Tachyon
HDFS

Marlin:基于Spark的高效的矩阵计算库

PasaLab / marlin

Unwatch 18 | Unstar 31 | Fork 12

A Distributed Matrix Operations Library Built on Top of Spark — Edit

80 commits | 4 branches | 0 releases | 4 contributors

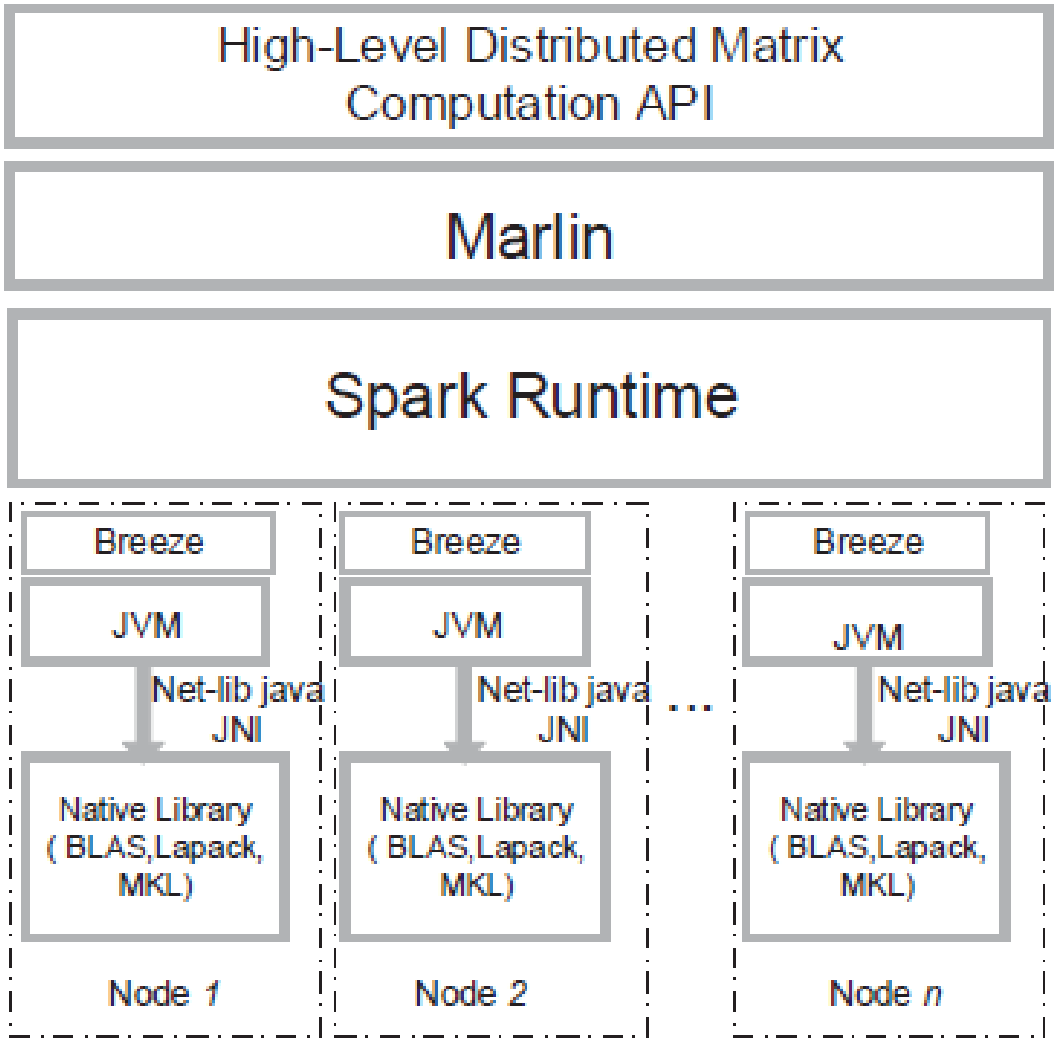
Branch: master | marlin / +

modify gitignore

Myasuka authored on 11 Apr — latest commit 702102d349

data	modify examples and data files	9 months ago
project	add project directory	7 months ago
src	add log info in Multiply	4 months ago

Marlin的系统架构图



矩阵的Marlin中的表示

- Local Matrix: 小规模矩阵且只存储在单机上。
- Broadcast Matrix: 小规模矩阵但在多个节点机上都有完整的存储。
- Distributed Matrix：大规模矩阵，需要切分分块存储在多个节点上。
  - Row Matrix: 按行切；
  - Block Matrix:按块切；

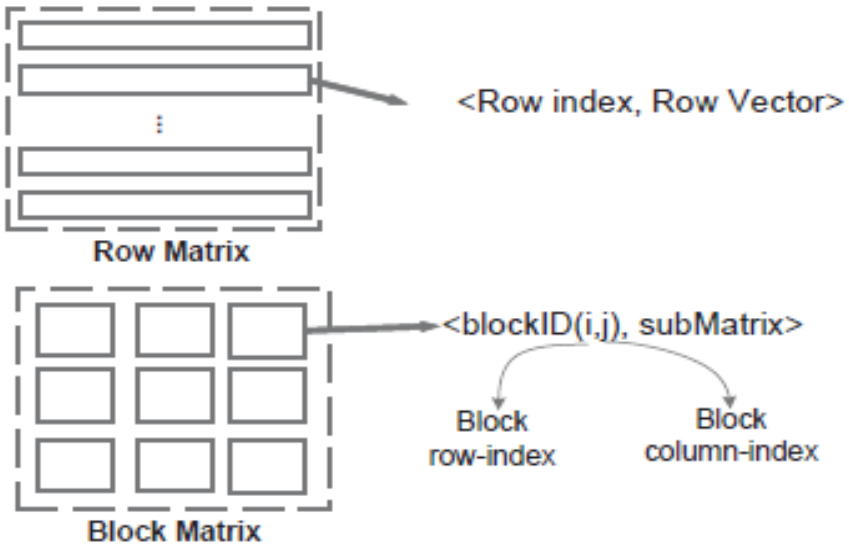
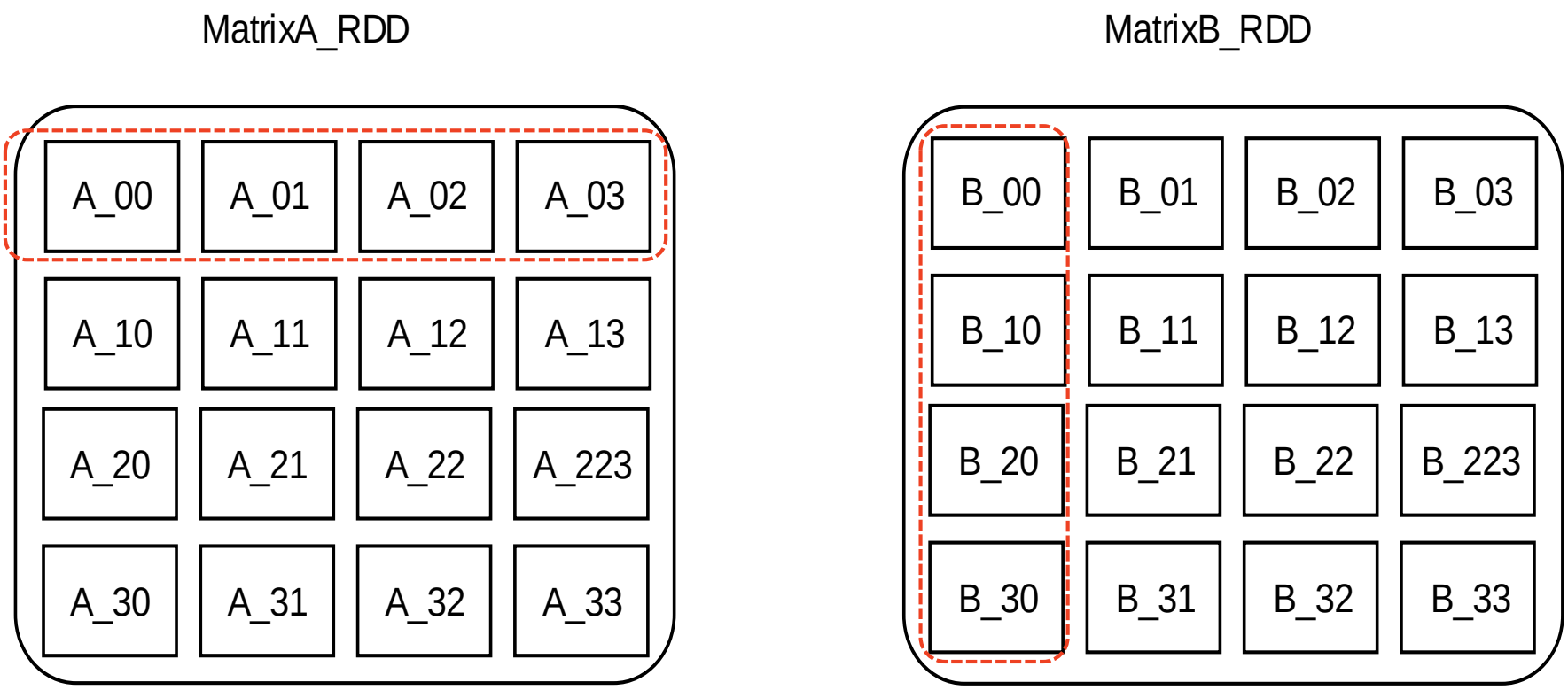
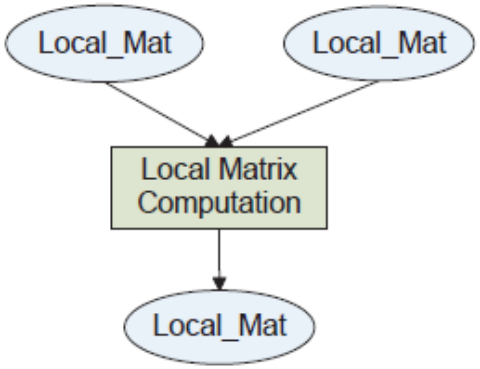


Figure 1: Distributed Matrix Representation

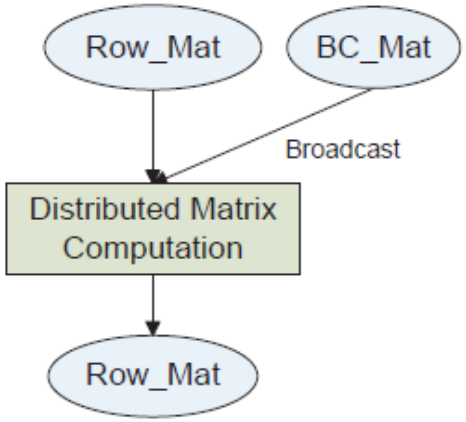
分布式矩阵相乘的基本策略



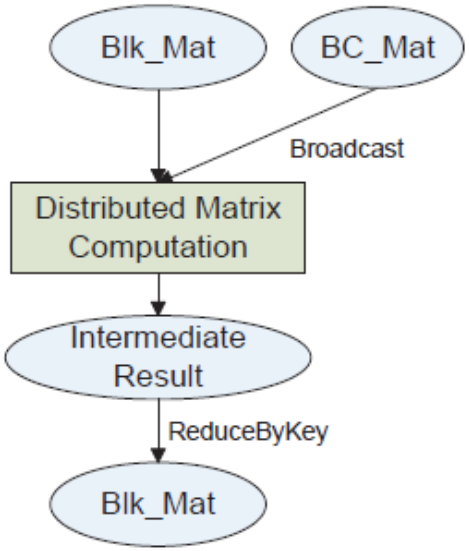
根据矩阵规模不同的Execution Graph



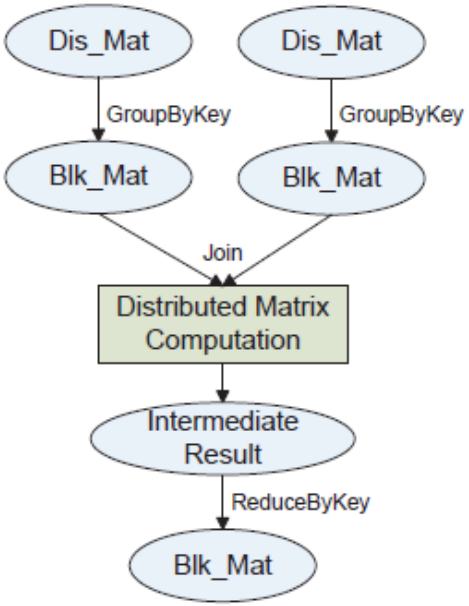
(a) *LocalMatrix * LocalMatrix*



(b) *RowMatrix * BroadcastMatrix*



(c) *BlockMatrix * BroadcastMatrix*



(d) *DistMatrix * DistMatrix*

对于分布式矩阵相乘不同的切分策略

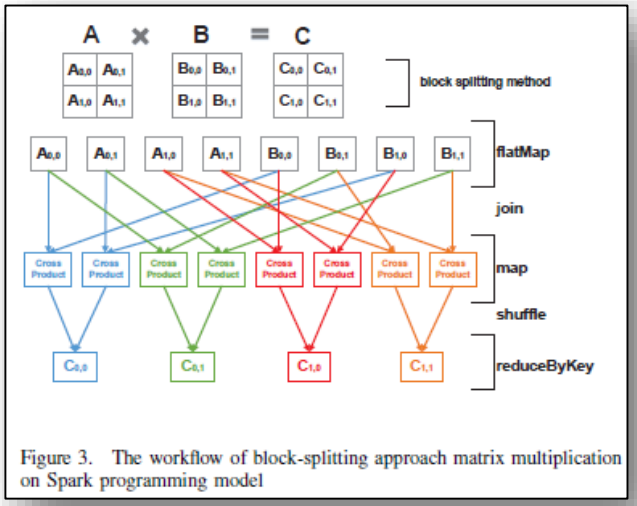


Figure 3. The workflow of block-splitting approach matrix multiplication on Spark programming model

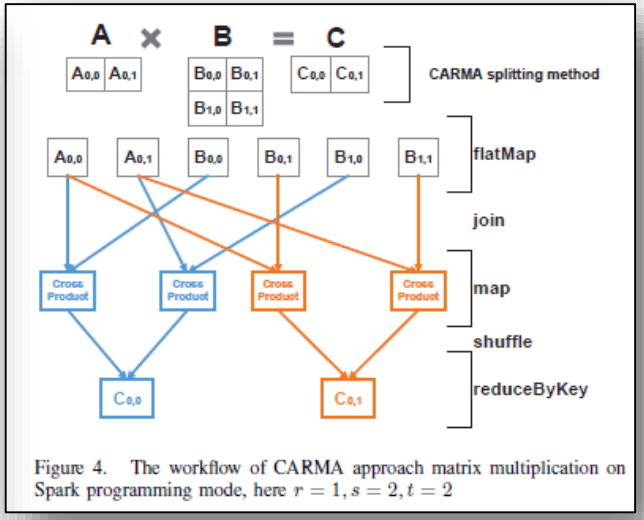


Figure 4. The workflow of CARMA approach matrix multiplication on Spark programming mode, here $r = 1, s = 2, t = 2$

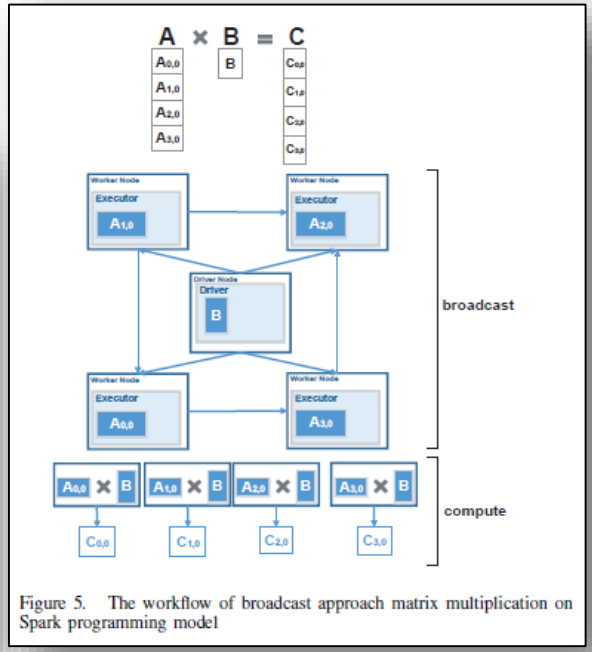


Figure 5. The workflow of broadcast approach matrix multiplication on Spark programming model

HAMA Blocking

CARMA Blocking

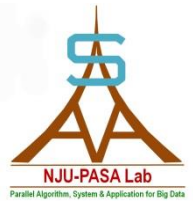
Broadcasting

流程的差异与性能的差异

Methods	Compt. Character		Small Scale Data (Time Cost Unit: sec)		
	# of BC	# of Shuffle	2k*2k*2k	4k*4k*4k	6k*6k*6k
Local_Mat * Local_Mat	0	0	3	16	53
Row_Mat * BC_Mat	1	0	5	12	16
Blk_Mat * BC_Mat	1	1	5	14	17
Row_Mat * Row_Mat	0	4	10	14	19
Blk_Mat * Row_Mat	0	4	12	16	21
Blk_Mat * Blk_Mat	0	4	12	17	23

Methods	Medium Scale Data (Time Cost Unit: sec)			Large Scale Data (Time Cost Unit: sec)		
	500k*1k*1k	500k*5k*1k	50k*1k*50k	20k*20k*20k	30k*30k*30k	40k*40k*40k
Local_Mat * Local_Mat	136	NA	NA	NA	NA	NA
Row_Mat * BC_Mat	9	30	40	205	NA	NA
Blk_Mat * BC_Mat	8	27	37	195	NA	NA
Row_Mat * Row_Mat	36	80	156	198	710	2569
Blk_Mat * Row_Mat	39	48	129	156	486	1620
Blk_Mat * Blk_Mat	40	46	120	162	522	1440

流程的差异与性能的差异



Methods	Compt. Character		Small Scale Data (Time Cost Unit: sec)		
	# of BC	# of Shuffle	2k*2k*2k	4k*4k*4k	6k*6k*6k
Local_Mat * Local_Mat	0	0	3	16	53
Row_Mat * BC_Mat	1	0	5	12	16
Blk_Mat * BC_Mat	1	1	5	14	17
Row_Mat * Row_Mat	0	4	10	14	19
Blk_Mat * Row_Mat	0	4	12	16	21
Blk_Mat * Blk_Mat	0	4	12	17	23

Methods	Medium Scale Data (Time Cost Unit: sec)			Large Scale Data (Time Cost Unit: sec)		
	500k*1k*1k	500k*5k*1k	50k*1k*50k	20k*20k*20k	30k*30k*30k	40k*40k*40k
Local_Mat * Local_Mat	136	NA	NA	NA	NA	NA
Row_Mat * BC_Mat	9	30	40	205	NA	NA
Blk_Mat * BC_Mat	8	27	37	195	NA	NA
Row_Mat * Row_Mat	36	80	156	198	710	2569
Blk_Mat * Row_Mat	39	48	129	156	486	1620
Blk_Mat * Blk_Mat	40	46	120	162	522	1440

流程的差异与性能的差异

NJU-PASA Lab
Parallel Algorithm, System & Application for Big Data

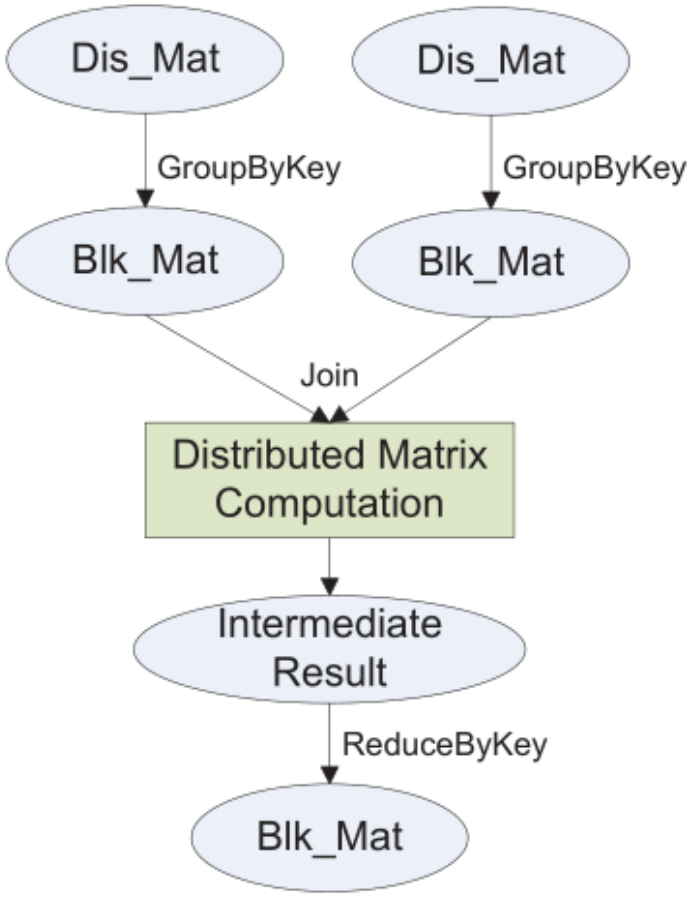
Methods	Compt. Character		Small Scale Data (Time Cost Unit: sec)		
	# of BC	# of Shuffle	2k*2k*2k	4k*4k*4k	6k*6k*6k
Local_Mat * Local_Mat	0	0	3	16	53
Row_Mat * BC_Mat	1	0	5	12	16
Blk_Mat * BC_Mat	1	1	5	14	17
Row_Mat * Row_Mat	0	4	10	14	19
Blk_Mat * Row_Mat	0	4	12	16	21
Blk_Mat * Blk_Mat	0	4	12	17	23

Methods	Medium Scale Data (Time Cost Unit: sec)			Large Scale Data (Time Cost Unit: sec)		
	500k*1k*1k	500k*5k*1k	50k*1k*50k	20k*20k*20k	30k*30k*30k	40k*40k*40k
Local_Mat * Local_Mat	136	NA	NA	NA	NA	NA
Row_Mat * BC_Mat	9	30	40	205	NA	NA
Blk_Mat * BC_Mat	8	27	37	195	NA	NA
Row_Mat * Row_Mat	36	80	156	198	710	2569
Blk_Mat * Row_Mat	39	48	129	156	486	1620
Blk_Mat * Blk_Mat	40	46	120	162	522	1440

大规模分布式矩阵相乘耗时分析

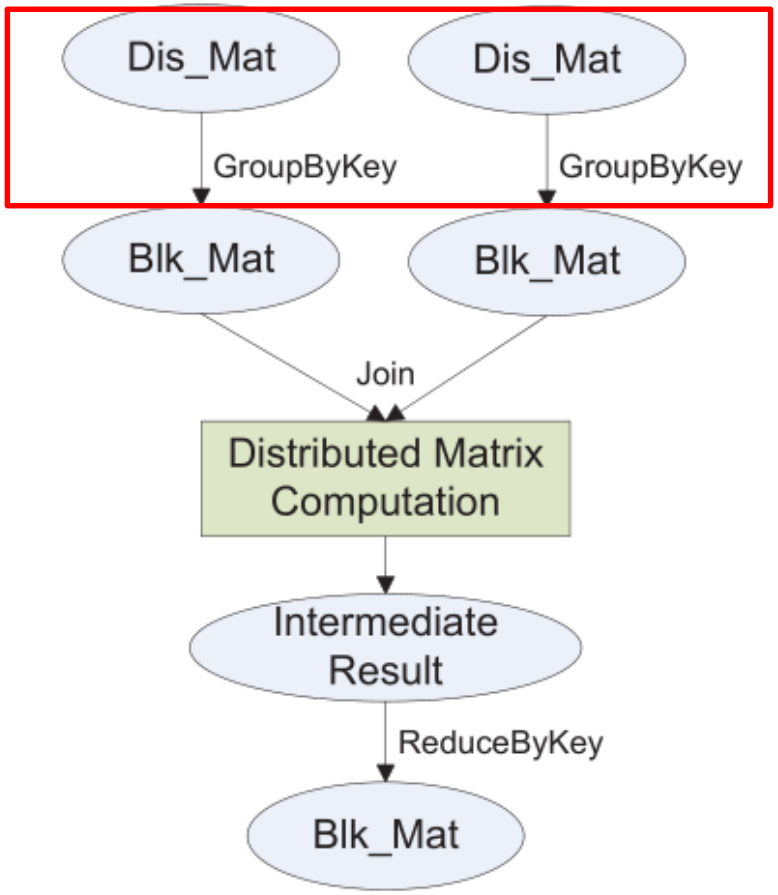
- T_total = T_getbestsplit + T_convertion + T_computation;

- 其中选择好的切分方式的耗时（T_getbestsplit）的计算是单机常数级的（几十毫秒就可以算好），耗时最多的主要还是分布式矩阵转换(T_convertion)和计算的时间(T_computation)。



(d) *DistMatrix * DistMatrix*

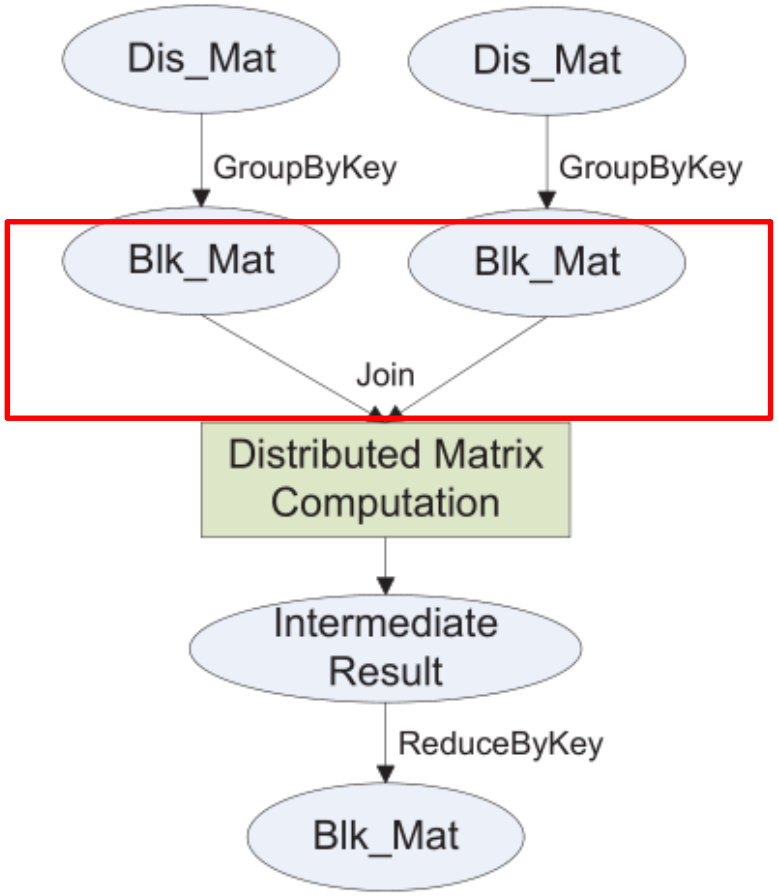
性能优化1:分布式矩阵在Spark上高效转换的优化



(d) *DistMatrix * DistMatrix*

- MLlib目前的做法是先将输入的分布式矩阵转成坐标矩阵，再讲坐标矩阵转成输出的分布式矩阵。
- Marlin做法通过计算边界，直接将分布式矩阵转成坐标矩阵转成所需的分布式矩阵。

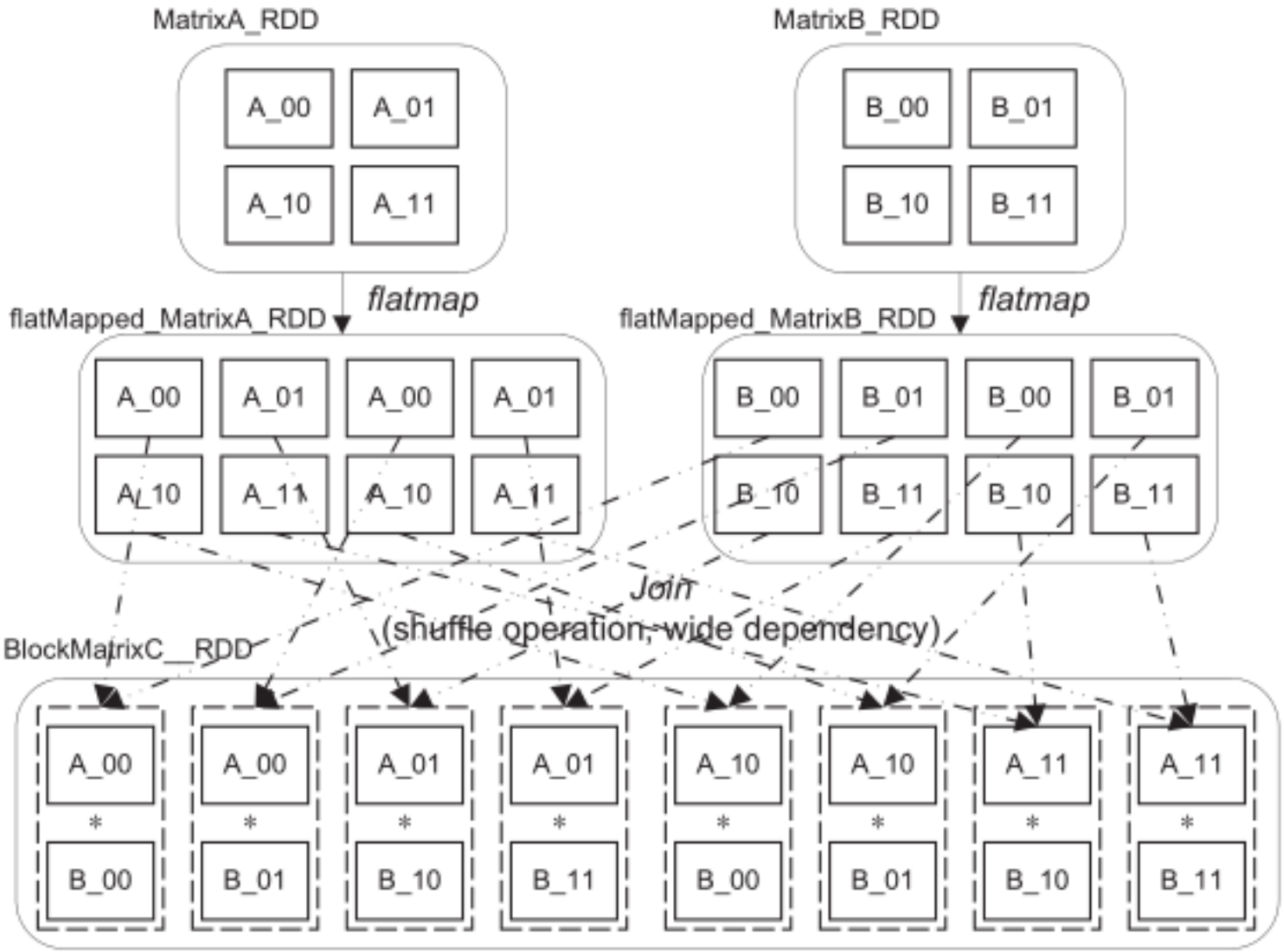
性能优化2：Join operation without disk-shuffling



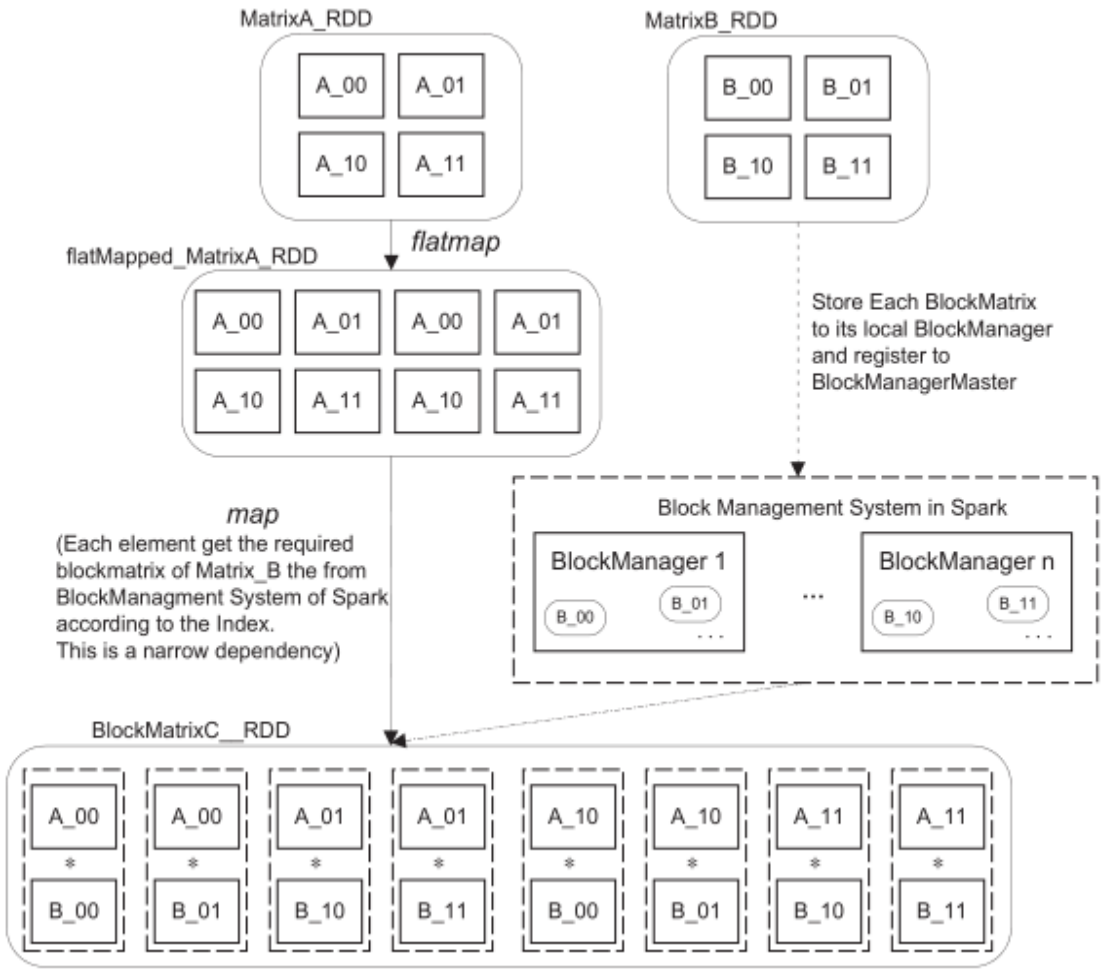
(d) *DistMatrix * DistMatrix*

- 两个切分好的分布式矩阵相乘时，需要通过Shuffle来将对应的子矩阵合并到同一台节点上，从而进行运算。
  - 我们能否提升这个shuffle的性能？

性能优化2：Join operation without disk-shuffling

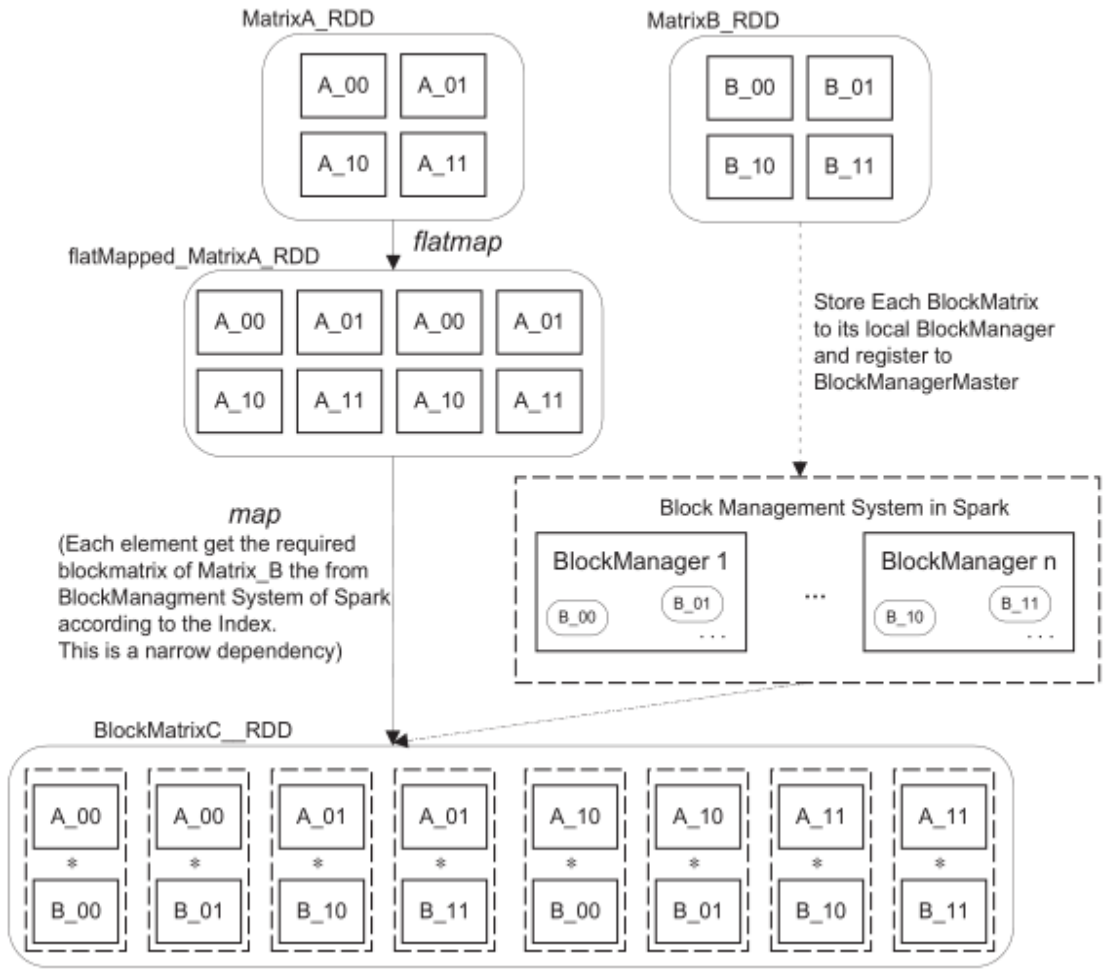


性能优化2：Join operation without disk-shuffling



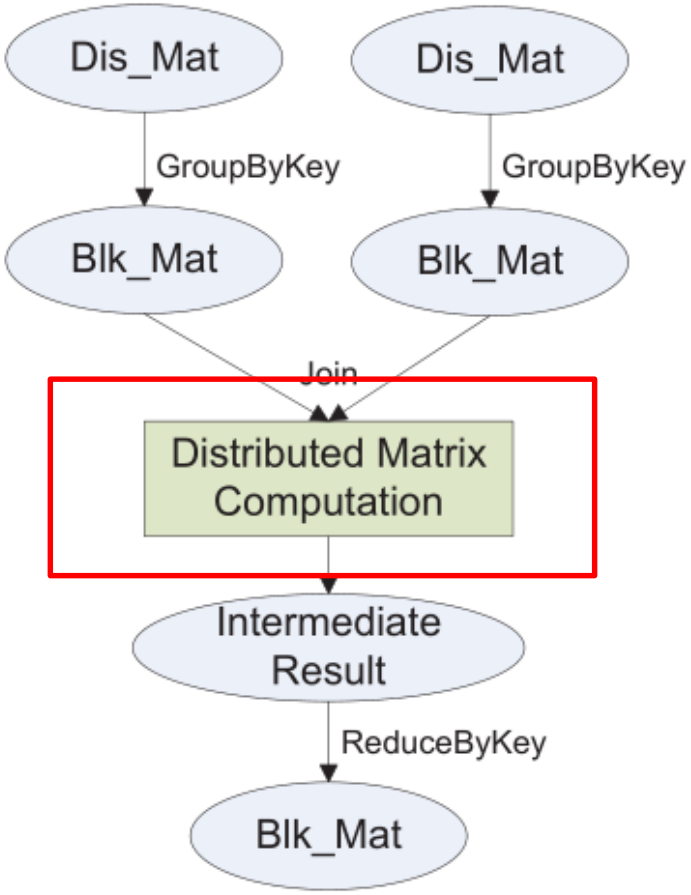
- 本操作的特点分析
  - 内存可以存储全部数据
  - 元素/块的依赖关系是可以预先算出的的，并不是全依赖

性能优化2：Join operation without disk-shuffling

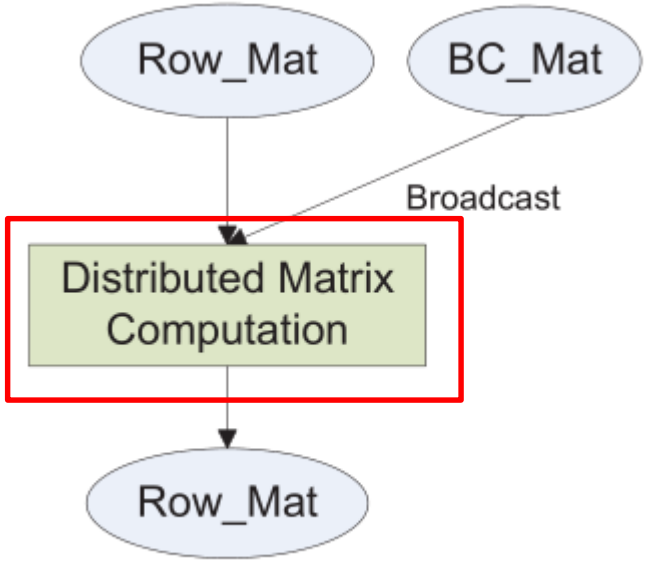


- 本操作的特点分析
  - 内存可以存储全部数据
  - 元素/块的依赖关系是可以预先算出的的，并不是全依赖
- 优化
  - 将其中一个块矩阵注册到BlockManager里面；另一个块矩阵的块去查找自己需要的块。
  - 将宽依赖变为窄依赖，同时避免了通过磁盘的shuffle

性能优化2：更高效的利用底层线性代数计算库

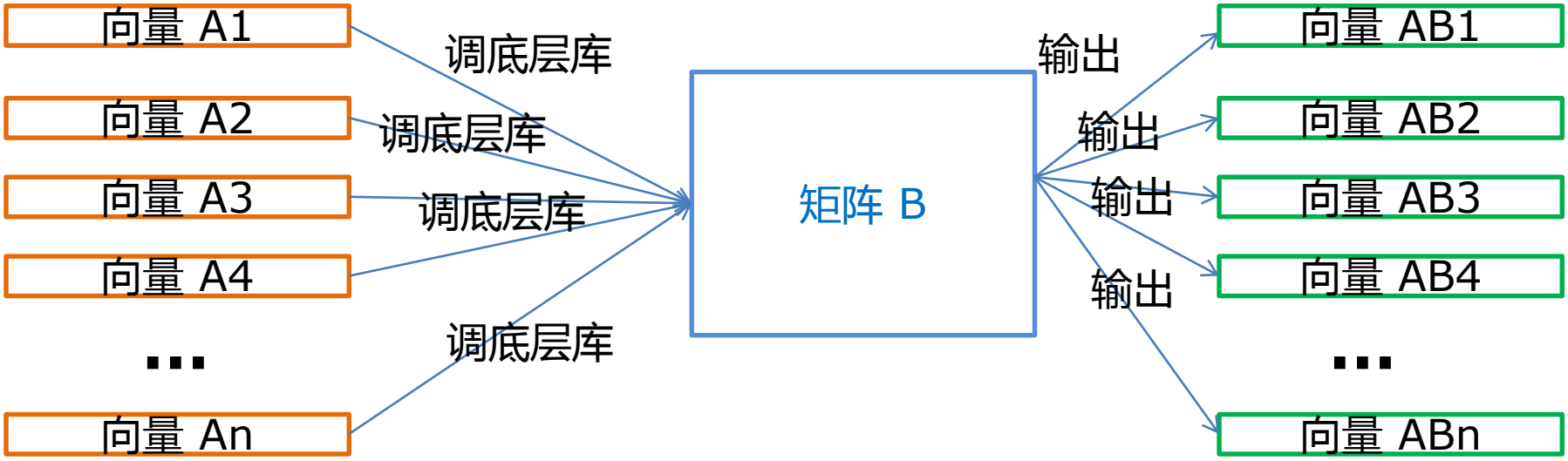


(d) *DistMatrix * DistMatrix*

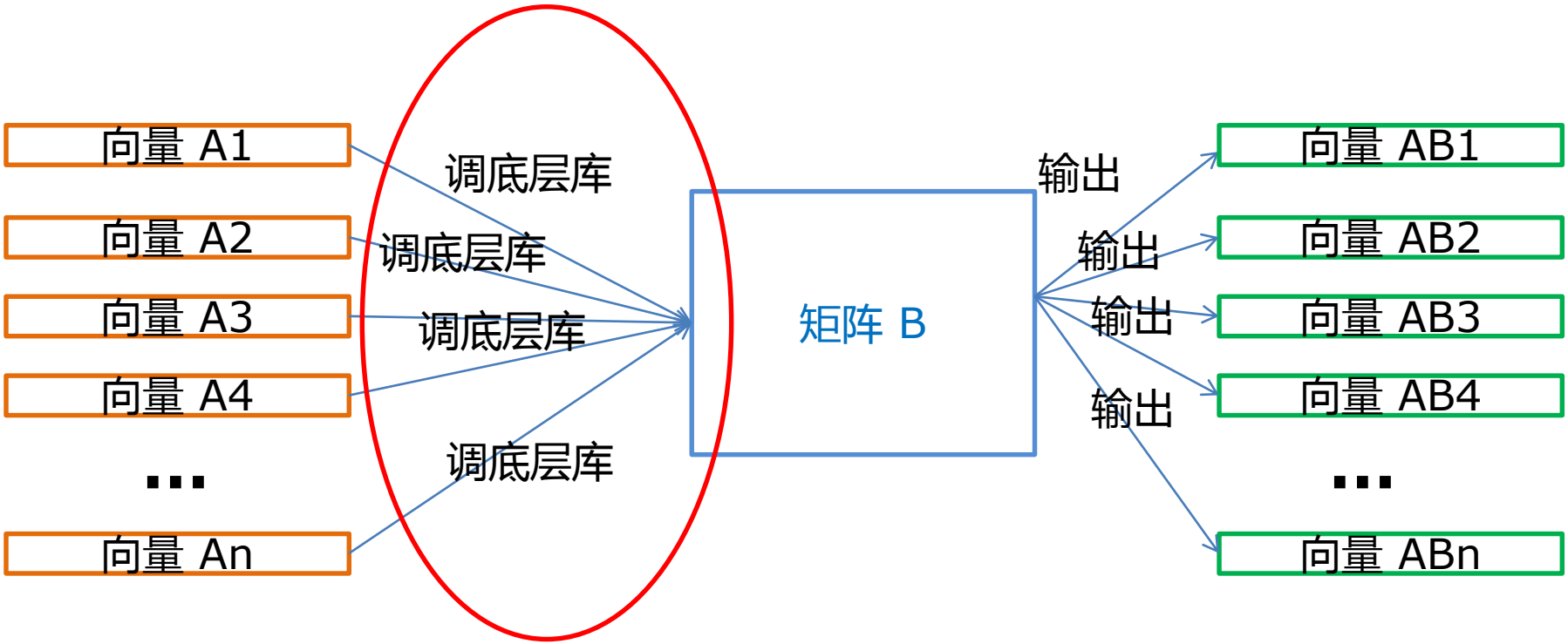


(b) *RowMatrix * BroadcastMatrix*

性能优化3：更高效的利用底层线性代数计算库

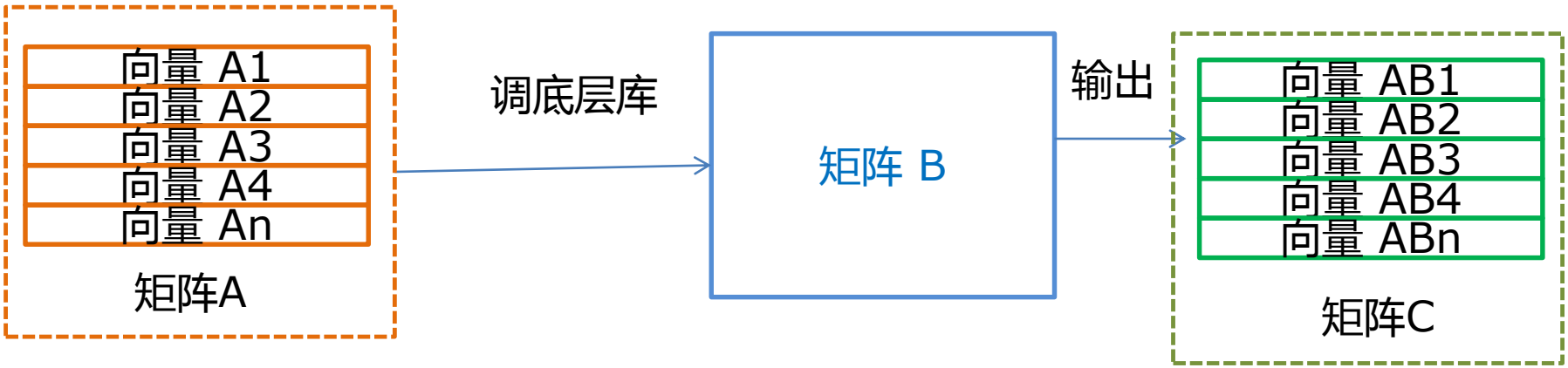


性能优化3：更高效的利用底层线性代数计算库



大量的系统调用Overhead!

性能优化3：更高效的利用底层线性代数计算库



整合一定量的数据进行批量操作，减少overhead！

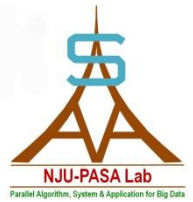
内容

- 研究动机与目标
- 大章鱼的编程模型与系统架构
- 大章鱼的系统实现与优化
- 大章鱼的性能评估
- Demo & 总结


NJU-PASA Lab
Parallel Algorithm, System & Application for Big Data

发布版省略

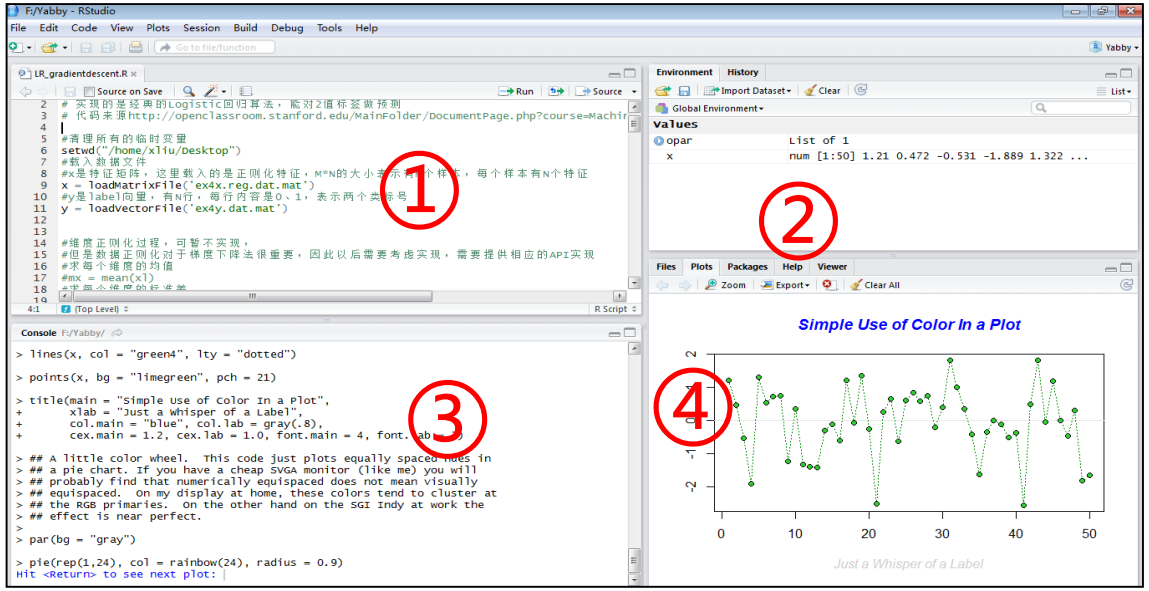
内容

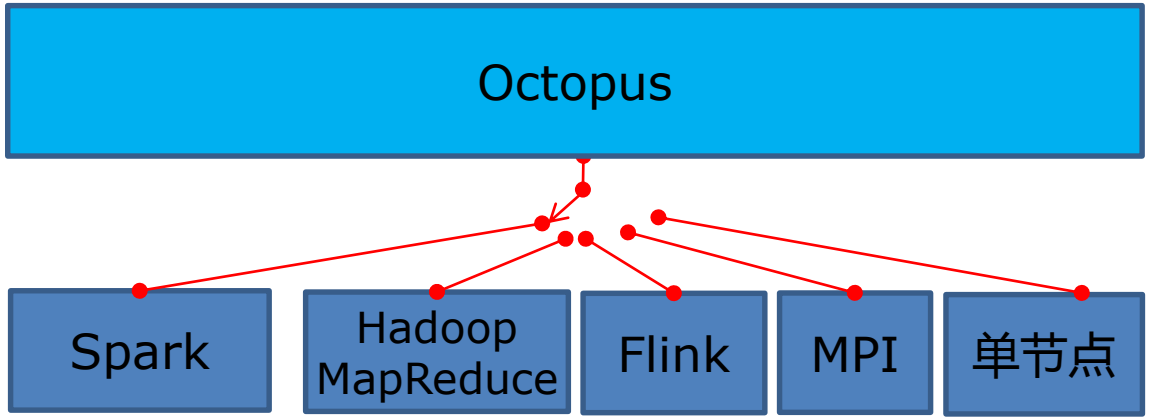


- 研究动机与目标
- 大章鱼的编程模型与系统架构
- 大章鱼的系统实现与优化
- 大章鱼的性能评估
- Demo & 总结

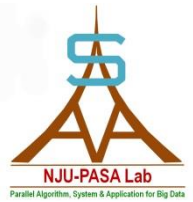
Octopus的工作方式总结

- 使用标准的R编程和开发环境，用户使用R语言、基于矩阵计算模型编写各种MLDM算法
- 已实现与Spark、MapReduce和MPI的集成，底层可无缝切换运行与不同的大数据平台
- 已实现LR、SVM、聚类、深度学习等一系列示范MLDM算法





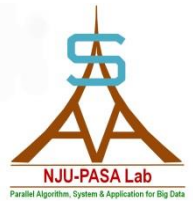
Octopus的特点总结



- 提供一个支持大规模矩阵操作的R编程接口，能与标准的R生态环境无缝结合；

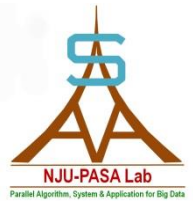
- 包含的分布式矩阵操作的接口都是高层的，无需用户具备分布式系统的基本概念或知识
  - 提供常见的分布式矩阵操作，包括矩阵相加、相乘、转置、多种分解算法等；
  - 支持对分布式矩阵的元素并行地执行R的标准函数和用户自定义函数的计算；（类似于单机R里面的lapply函数）

Octopus的特点总结



- 底层支持选用多种分布式计算框架和存储平台，能与现有的大数据生态系统无缝对接
  - 支持的分布式计算引擎包括Spark, MapReduce, MPI，同时也提供单机的R计算引擎；
  - 数据可以存储在HDFS、Tachyon上，并可以通过Tachyon在多个计算引擎间快速共享矩阵数据；
- 同一套程序可以无需修改地运行在多个底层计算平台上，从而减轻用户开发和维护代码的负担。
  - 支持对底层计算引擎的灵活配置、智能选择；
  - 避免在各个计算平台间迁移重实现application的成本；

与Spark生态系统的交互整合



- 为什么需要整合？

与Spark生态系统的交互整合

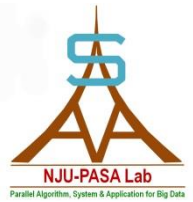
- 为什么需要整合？
  - 与Spark系统的多个组件整合成更强大的处理流水线
  - 各个模块发挥好自己的优势
    - 大章鱼的优势在于矩阵处理和跨平台计算

与Spark生态系统的交互整合

- 为什么需要整合？
  - 与Spark系统的多个组件整合成更强大的处理流水线
  - 各个模块发挥好自己的优势
    - 大章鱼的优势在于矩阵处理和跨平台计算

- 通过Spark 1.4里的DataFrame进行整合
  - DataFrame已经被广泛应用于Spark的多个组件, Spark SQL, Spark ML, SparkR等。
  - 大章鱼系统提供两种方式将系统内的分布式矩阵OctMatrix转成Spark里面的DataFrame
    - 通过直接RDD转换（适用于执行引擎是Spark）
    - 将OctMatrix看成是DataFrame的一个外部数据源，类似于Jason, Parquet（适用于任何情况）

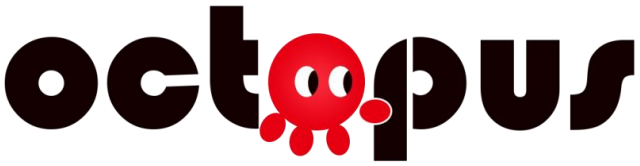
下一步工作（部分）



- 更多高层编程语言的支持
  - Python、Julia等
- 运行时多引擎的自动选择
  - 根据处理的矩阵规模、操作类型，结合集群资源情况，自动地选择最合理的底层引擎进行计算。
- 更多并行化计算模型的整合

NJU-PASA Lab
Parallel Algorithm, System & Application for Big Data

The End & Thank you!



项目主页： http://pasa-bigdata.nju.edu.cn/octopus/

个人微博：顾荣_NJU
电子邮箱：gurongwalker@gmail.com