火龙果•整理
uml.org.cn

背景

- 虽然MapReduce已经较有效，但主要是建立非循环的数据流模型，不能满足用户更大的需求
  - Iterative jobs：很多机器学习算法，需要在多个并行操作之间重用数据集，MapReduce job 中，为容错将数据存在磁盘中并复制，需要很多I/O操作，性能代价很高
  - Interactive analytics：Hadoop经常用于专用的探索性查询，用SQL接口，如：Pig，Hive。而理想情况下，用户可能想拷贝一个数据集到内存中，多次进行查询。但是，实际上Hadoop有很大的延迟，因为要从磁盘读取数据
  - Real-time：MapRedeuce的延迟很大，不能支持实时流处理

火龙果•整理
uml.org.cn

背景

- 解决方法：为这些应用提供特定的模型
  - Google Pregel：用于图处理
  - Iterative Mapreduce：处理迭代任务
  - Storm for Stream ：处理实时流数据
- 问题：
  - 不能覆盖所有的用例
  - 这些单个的应用，无法进行组合
- 这些应用都需要的是：有效的数据共享原语。而这个是MapReduce所缺少的

火龙果•整理
uml.org.cn

背景

- 如果不用复制到磁盘的方式，如何处理容错？
  - 需要在网络中拷贝数据，实现容错
  - 比内存读写慢10-100倍
- 是否可以不复制，实现容错？

火龙果•整理
uml.org.cn

目录

- 背景
- Spark
  - 概述
  - RDD
  - 两个共享变量：广播变量、累加器
  - 调度机制
  - Spark性能实验
- Spark Streaming
- 应用实例
  - 淘宝使用Spark

火龙果•整理
uml.org.cn

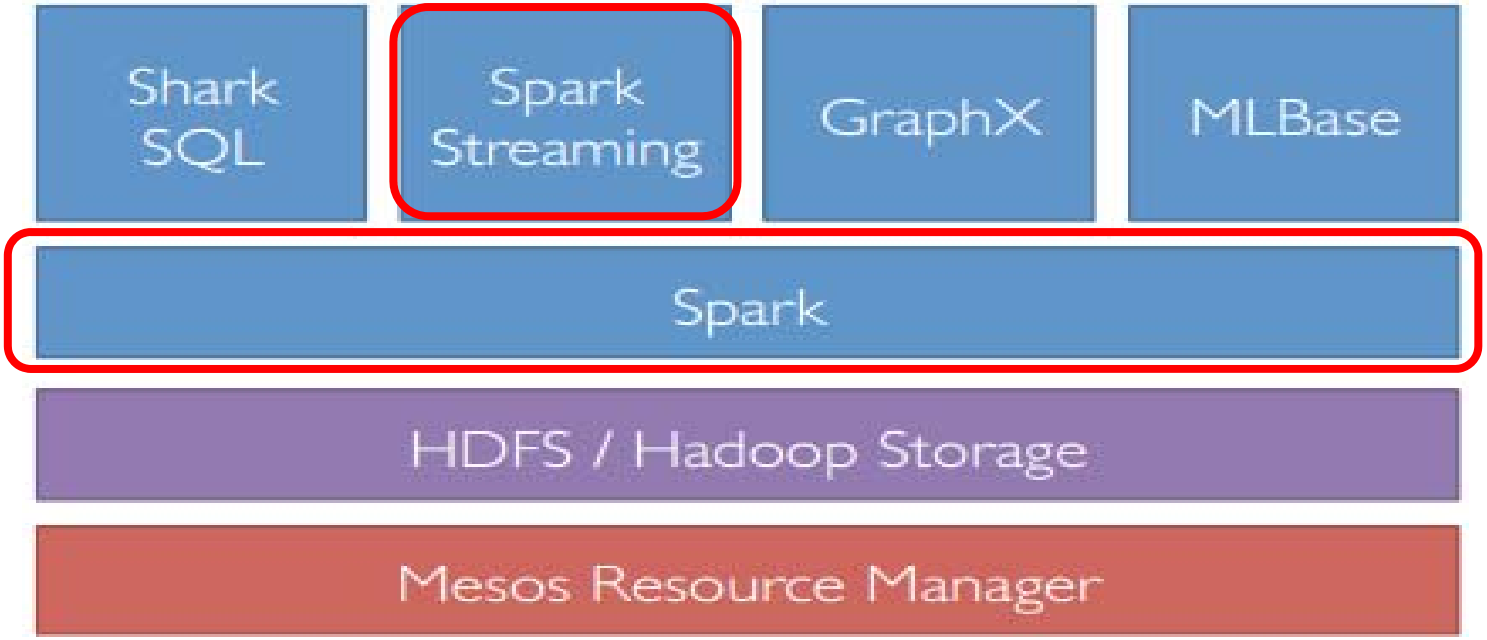
Spark目录

- Spark概述
- RDD
- 两个共享变量：广播变量、累加器
- 调度机制
- Spark性能实验

火龙果•整理
uml.org.cn

背景

- UC Berkeley AMP实验室数据分析栈



火龙果•整理
uml.org.cn

Spark

- **定义：** 一个开源、高效的分布式并行计算系统
- **目标：** 是支持迭代的算法、交互式数据分析
- **思想：** 内存集群计算——可在内存集群计算中将数据集缓存在内存中，以缩短访问延迟
- **实现：** 由Scala语言实现
  - 优势：可以直接兼容JVM、语言风格简洁、支持闭包
  - 同时支持Java、Python API
- **适用性**
  - 适用于批处理应用，数据集上所有数据都有相同的操作
  - 不适用于异步细粒度的更新状态，如：web应用的存储系统或增量式的网页爬虫

火龙果•整理
uml.org.cn

Spark

- 性能超Hadoop百倍；将中间结果存在内存中，对于迭代操作效率很高
- 提供比Hadoop更上层的API，算法实现代码量仅有其1/10或1/100
- 比Hadoop更通用，不仅仅有Map、Reduce两种操作
- 兼容Hadoop存储API，支持读写所有Hadoop支持的系统，如HDFS、Hbase、SequenceFiles……

火龙果•整理
uml.org.cn

Spark目录

- Spark概述
- RDD——不用复制实现容错
- 两个共享变量：广播变量、累加器
- 调度机制
- Spark性能实验

火龙果•整理
uml.org.cn

RDD-定义

- RDD:Resilient Distributed Datasets（弹性分布式数据集）
- 只读、可分区的数据集（数据格式不限，如：数组、对象等）
- 只可以粗粒度，由确定的操作创建

火龙果•整理
uml.org.cn

RDD-创建

- 稳定的存储数据
  - 例如：读取共享文件系统（如HDFS）的文件
  - 将一个Scala集合（如：数组）进行并行化，划分成多个片并发送到多个节点上
- 转变已有的RDD
  - 通过给定的函数操作，进行转变
- *Transformations* operations: map, filter, join, etc.
- RDD都是Lazy的， Transformations的操作并不实际发生，只有当第一个运算操作执行时，才流水线的进行Transformations。

火龙果•整理
uml.org.cn

RDD-创建

- Persistence：可以指定哪些RDD会被重用，并选择存储策略，例如在内存中保存。
- Partitioning：可以指定RDD基于某种方式在分布式节点上进行划分。例如两个数据集做join操作，可以通过对数据集的哈希划分实现优化。

火龙果•整理
uml.org.cn

RDD-存储

- 划分用hash划分的方式，也可以指定划分方式
- 指定持久化
  - Persist：会保存在内存中
  - Cache：只是暗示作用，但是若内存不够，则不会存到内存中
- 可指定存储内存的优先级：MEMORY_ONLY、DISK_ONLY 、DISK_ONLY_2 、MEMORY_ONLY_2、 MEMORY_ONLY_SER、MEMORY_AND_DISK……
  - 默认持久化级别为：MEMORY_ONLY
  - 目前存储级别一旦确定就不能改变
  - 可以用unpersist来删除cache的数据集
- 当内存不够时，会根据LRU替换出RDD到磁盘中

火龙果•整理
uml.org.cn

RDD-并行操作

- Action——返回值到应用中，或输出数据到存储系统，例如：count、collect、save
- Transformation——特性：lazy，pipeline。例如：map、filter、join

Transformations	$map(f : T \Rightarrow U)$	:	$RDD[T] \Rightarrow RDD[U]$
	$filter(f : T \Rightarrow Bool)$	:	$RDD[T] \Rightarrow RDD[T]$
	$flatMap(f : T \Rightarrow Seq[U])$	:	$RDD[T] \Rightarrow RDD[U]$
	$sample(fraction : Float)$	:	$RDD[T] \Rightarrow RDD[T]$ (Deterministic sampling)
	$groupByKey()$	:	$RDD[(K, V)] \Rightarrow RDD[(K, Seq[V])]$
	$reduceByKey(f : (V, V) \Rightarrow V)$	:	$RDD[(K, V)] \Rightarrow RDD[(K, V)]$
	$union()$	:	$(RDD[T], RDD[T]) \Rightarrow RDD[T]$
	$join()$	:	$(RDD[(K, V)], RDD[(K, W)]) \Rightarrow RDD[(K, (V, W))]$
	$cogroup()$	:	$(RDD[(K, V)], RDD[(K, W)]) \Rightarrow RDD[(K, (Seq[V], Seq[W]))]$
	$crossProduct()$	:	$(RDD[T], RDD[U]) \Rightarrow RDD[(T, U)]$
	$mapValues(f : V \Rightarrow W)$	:	$RDD[(K, V)] \Rightarrow RDD[(K, W)]$ (Preserves partitioning)
	$sort(c : Comparator[K])$	:	$RDD[(K, V)] \Rightarrow RDD[(K, V)]$
	$partitionBy(p : Partitioner[K])$	:	$RDD[(K, V)] \Rightarrow RDD[(K, V)]$
Actions	$count()$	:	$RDD[T] \Rightarrow Long$
	$collect()$	:	$RDD[T] \Rightarrow Seq[T]$
	$reduce(f : (T, T) \Rightarrow T)$	:	$RDD[T] \Rightarrow T$
	$lookup(k : K)$	:	$RDD[(K, V)] \Rightarrow Seq[V]$ (On hash/range partitioned RDDs)
	$save(path : String)$	:	Outputs RDD to a storage system, *e.g.*, HDFS

火龙果•整理
uml.org.cn

例子

- 对HDFS上的log文件进行分析，统计其中出现“error”消息的log数量

- lines = spark.textFile("hdfs://...")
- errors = lines.filter(_.startsWith("ERROR"))
- errors.persist()

- errors.count()

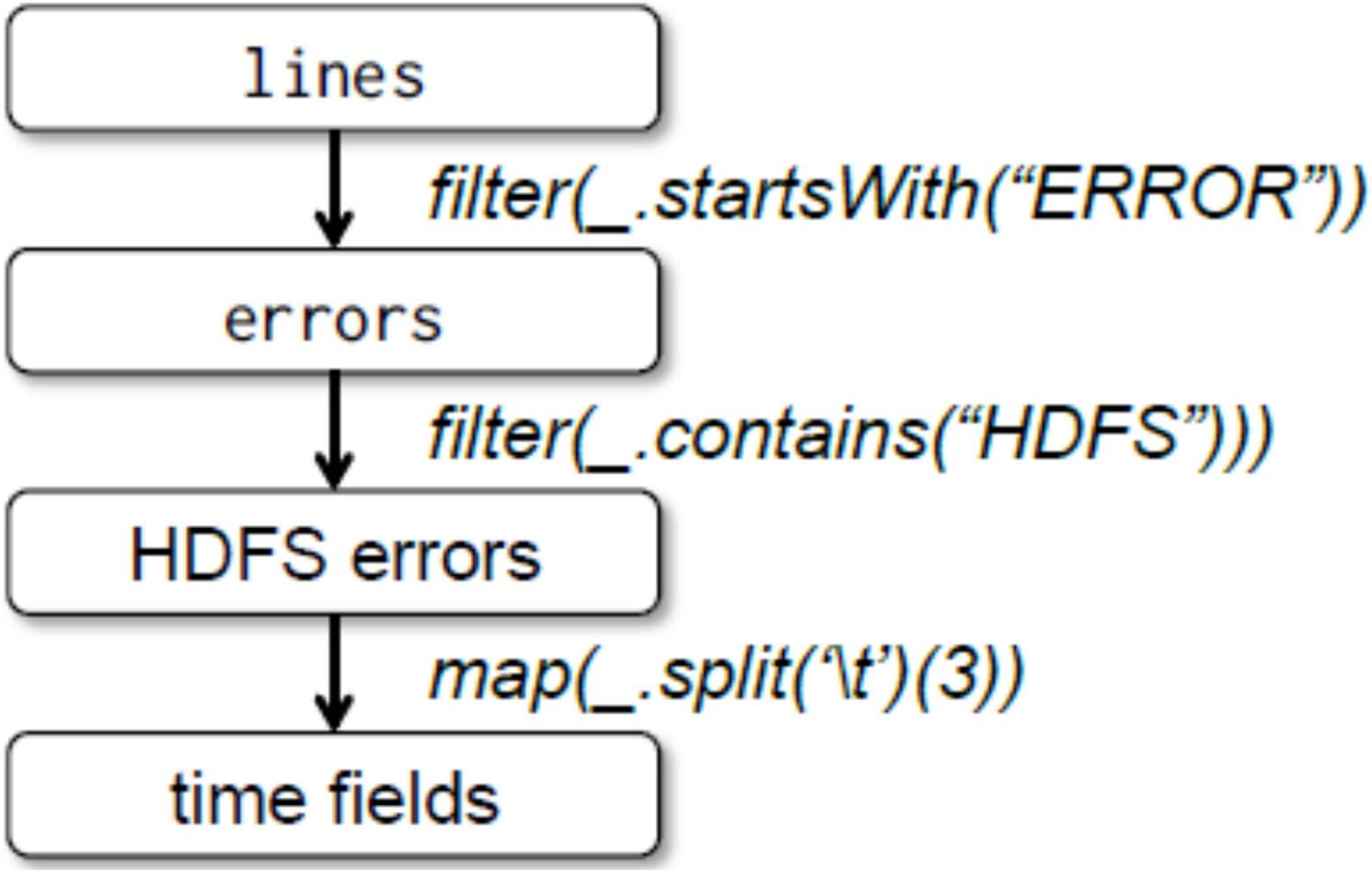
火龙果•整理
uml.org.cn

例子

- // Count errors mentioning MySQL:
- errors.filter(_.contains("MySQL")).count()
- // Return the time fields of errors mentioning
- // HDFS as an array (assuming time is field
- // number 3 in a tab-separated format):
- errors.filter(_.contains("HDFS")).map(_.split('\t')(3))
  .collect()

火龙果•整理
uml.org.cn

Lineage



举例

- Action&transformation举例

```
scala> val file=sc.textFile("/home/aloe/test")
14/03/09 15:32:25 INFO MemoryStore: ensureFreeSpace(31647) called with curMem=
126556, maxMem=308713881
14/03/09 15:32:25 INFO MemoryStore: Block broadcast_4 stored as values to memo
ry (estimated size 30.9 KB, free 294.3 MB)
file: org.apache.spark.rdd.RDD[String] = MappedRDD[17] at textFile at <console
>:12

scala> val count=file.flatMap(line=>line.split(" ")).map(word=>(word,1)).reduc
eByKey(_+_)
14/03/09 15:32:39 INFO FileInputFormat: Total input paths to process : 1
count: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[22] at reduc
eByKey at <console>:14

scala> count.collect()
14/03/09 15:33:08 INFO SparkContext: Starting job: collect at <console>:17
14/03/09 15:33:08 INFO DAGScheduler: Registering RDD 20 (reduceByKey at <conso
le>:14)
14/03/09 15:33:09 INFO SparkContext: Job finished: collect at <console>:17, to
ok 0.491183397 s
res0: Array[(String, Int)] = Array((links:,1), (be,1), (predicates,1), (true),
1), (example,1), (edge,2), (takes,1), (number,1), (situations,1), (operator,2)
, (only,1), (restrict,1), (returns,1), (edges,2), (can,1), (eliminate,1), (gra
ph,2), (subgraph,2), (remove,1), (broken,2), (to,3), (vertex,3), (used,1), (co
ntaining,1), ((evaluate,1), (For,1), (following,1), (of,2), (The,2), (interest
,1), (we,1), (the,8), (predicate,2), (in,2), (or,1), (predicate.,1), (and,5),
(links.,1), (code,1), (connect,1), (satisfy,3), (that,3), (vertices,3))
```

火龙果•整理
uml.org.cn

举例

```
class Mapper
    method Map(docid a, doc d)
        for all term t ∈ doc d do
            Emit(term t, count 1)

class Reducer
    method Reduce(term t, counts [c1, c2, ...])
        sum ← 0
        for all count c ∈ counts [c1, c2, ...] do
            sum ← sum + c
        Emit(term t, count s)
```

火龙果•整理
uml.org.cn

RDD的优点

Aspect	RDDs	Distr. Shared Mem.
Reads	Coarse- or fine-grained	Fine-grained
Writes	Coarse-grained	Fine-grained
Consistency	Trivial (immutable)	Up to app / runtime
Fault recovery	Fine-grained and low-overhead using lineage	Requires checkpoints and program rollback
Straggler mitigation	Possible using backup tasks	Difficult
Work placement	Automatic based on data locality	Up to app (runtimes aim for transparency)
Behavior if not enough RAM	Similar to existing data flow systems	Poor performance (swapping?)

火龙果•整理
uml.org.cn

RDD不适合的事情

- Spark是一种粗粒度数据并行
- Spark无法完美支持细粒度、异步更新的操作。图计算就有此类操作，所以此时Spark不如GraphLab（一个大规模图计算框架）；还有一些应用， 需要细粒度的日志更新和数据检查点，它也不如RAMCloud（斯坦福的内存存储和计算研究项目）

火龙果•整理
uml.org.cn

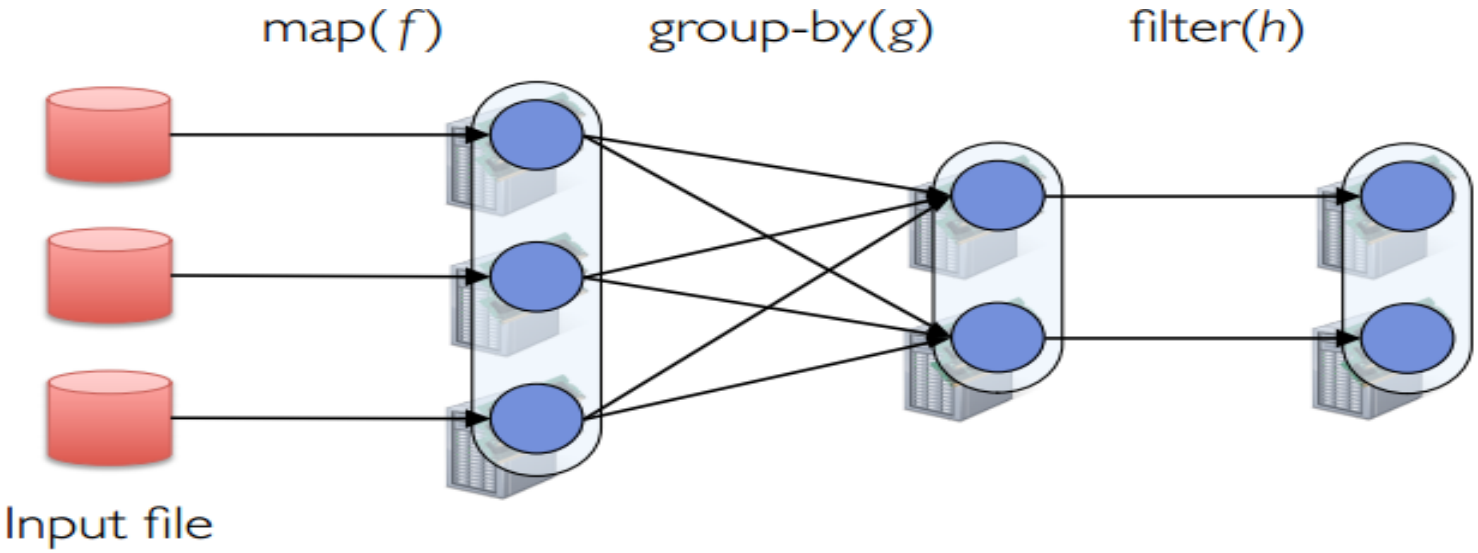
RDD-容错

- 基于Lineage
- 可指定Checkpoint

火龙果•整理
uml.org.cn

RDD-容错—Lineage

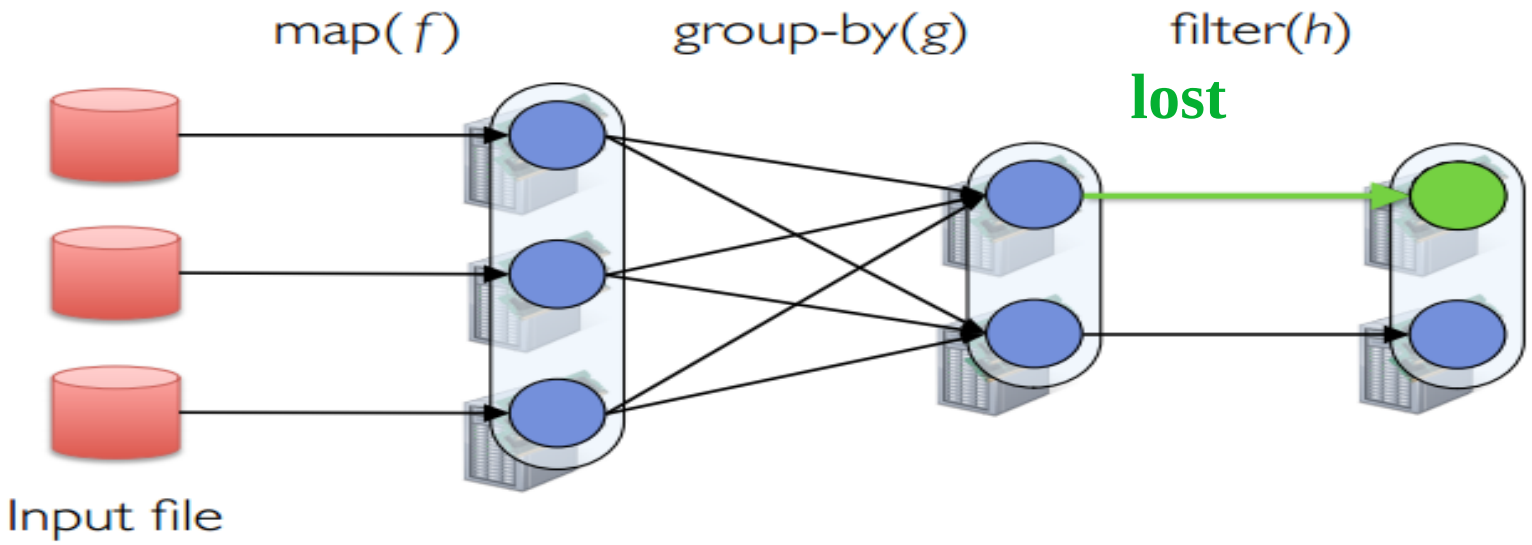
- Lineage图



火龙果•整理
uml.org.cn

RDD-容错—Lineage

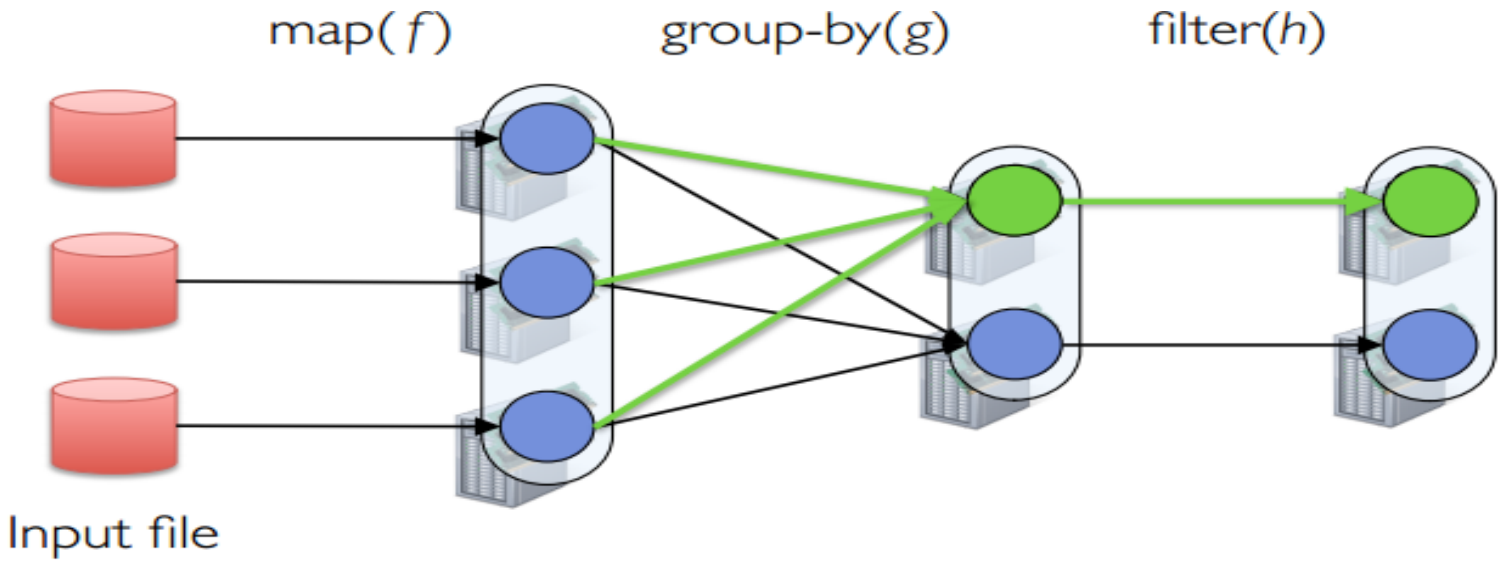
- 假设丢失了一个数据集



火龙果•整理
uml.org.cn

RDD-容错

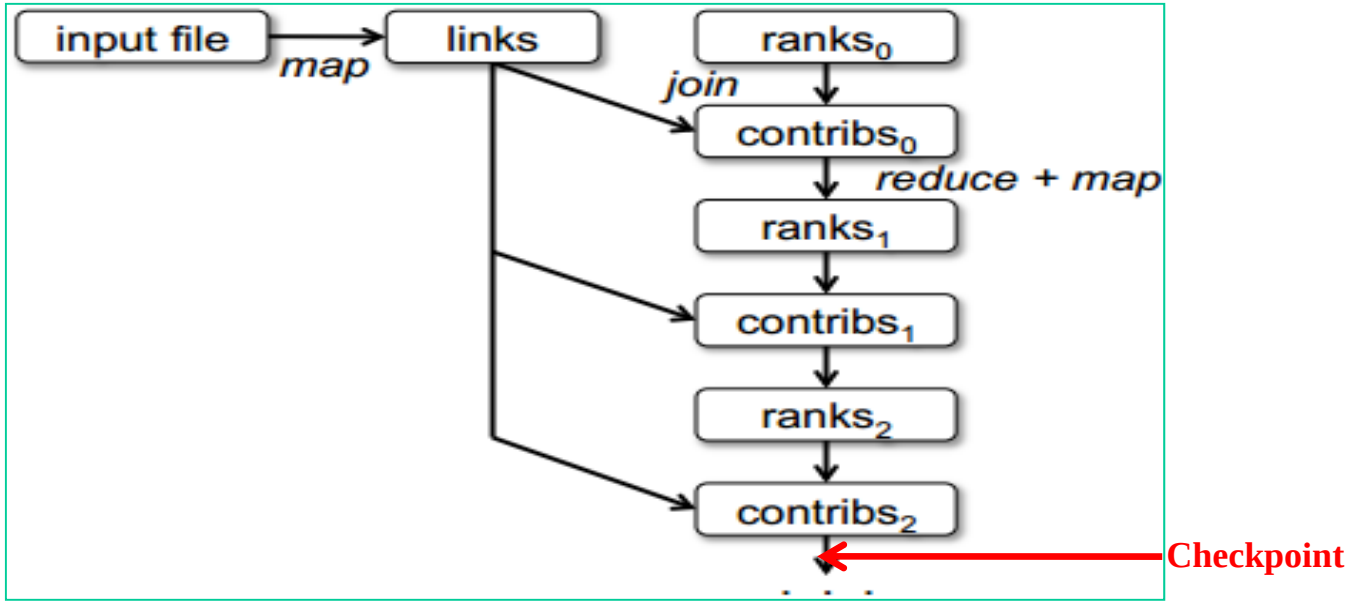
- 回溯整个Lineage图，利用上层的数据，重新构造丢失的RDD数据集



火龙果•整理
uml.org.cn

RDD-容错

- Checkpoint



- 在需要时指定，例如：PageRank中迭代次数很多，则可以在指定时间进行checkpoint。便于快速恢复

火龙果•整理
uml.org.cn

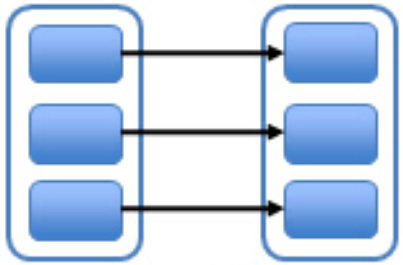
RDD-容错

- straggler掉队者
  - 推测式执行：当发现某台机器运行很慢的时候，在另一个机器上，并行的重新执行该任务，取先执行完的结果
  - 在实际使用中，会记录straggler黑名单，即记住那些掉队者，在分配任务时，尽量不考虑它们

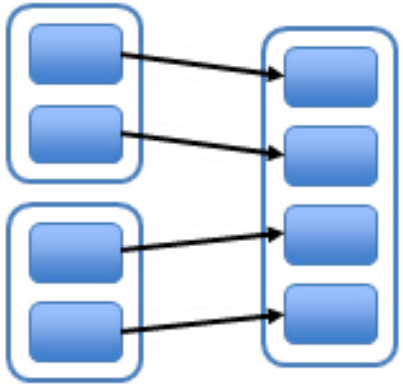
火龙果•整理
uml.org.cn

RDD依赖关系

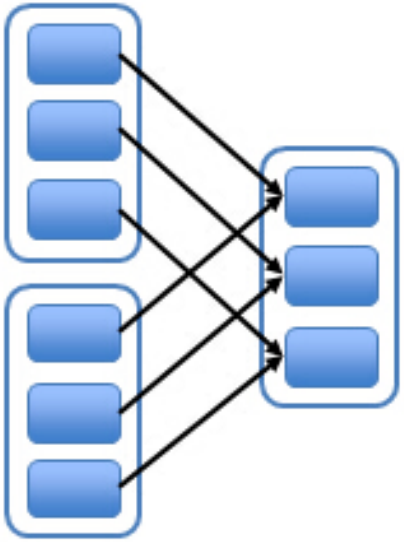
"Narrow" deps:



map, filter

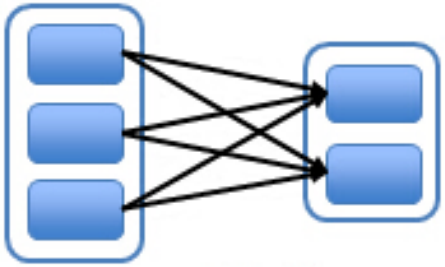


union

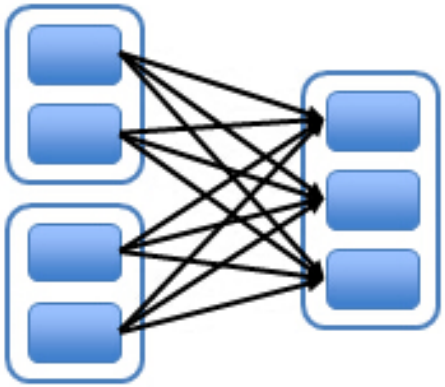


join with inputs co-partitioned

"Wide" (shuffle) deps:



groupByKey



join with inputs not co-partitioned

火龙果•整理
uml.org.cn

RDD表示

Operation	Meaning
partitions()	Return a list of Partition objects
preferredLocations(*p*)	List nodes where partition *p* can be accessed faster due to data locality
dependencies()	Return a list of dependencies
iterator(*p*, *parentIters*)	Compute the elements of partition *p* given iterators for its parent partitions
partitioner()	Return metadata specifying whether the RDD is hash/range partitioned

火龙果•整理
uml.org.cn

Spark目录

- Spark概述
- RDD——不用复制实现容错
- 两个共享变量：广播变量、累加器
- 调度机制
- Spark性能实验

火龙果•整理
uml.org.cn

广播变量

- 在并行操作中，将一个只读数据块，分发到每个机器上，而不是每个task中
- 利用Broacast variable包裹值，保证每个节点只拷贝一次

火龙果•整理
uml.org.cn

累加器

- 可用于计数，或sum
  - 可以是任意类型
  - 但要有add操作和zero值
  - 只有驱动程序可读；集群可以用+=，不可读
- 在Worker上建立一个单独的副本累加器，作为每个运行任务的线程的本地变量，从0开始。
- 任务运行结束后，发送信息到驱动程序，包括每个累加器的更新，驱动程序对每个操作的每个分区进行一次更新，避免重复执行

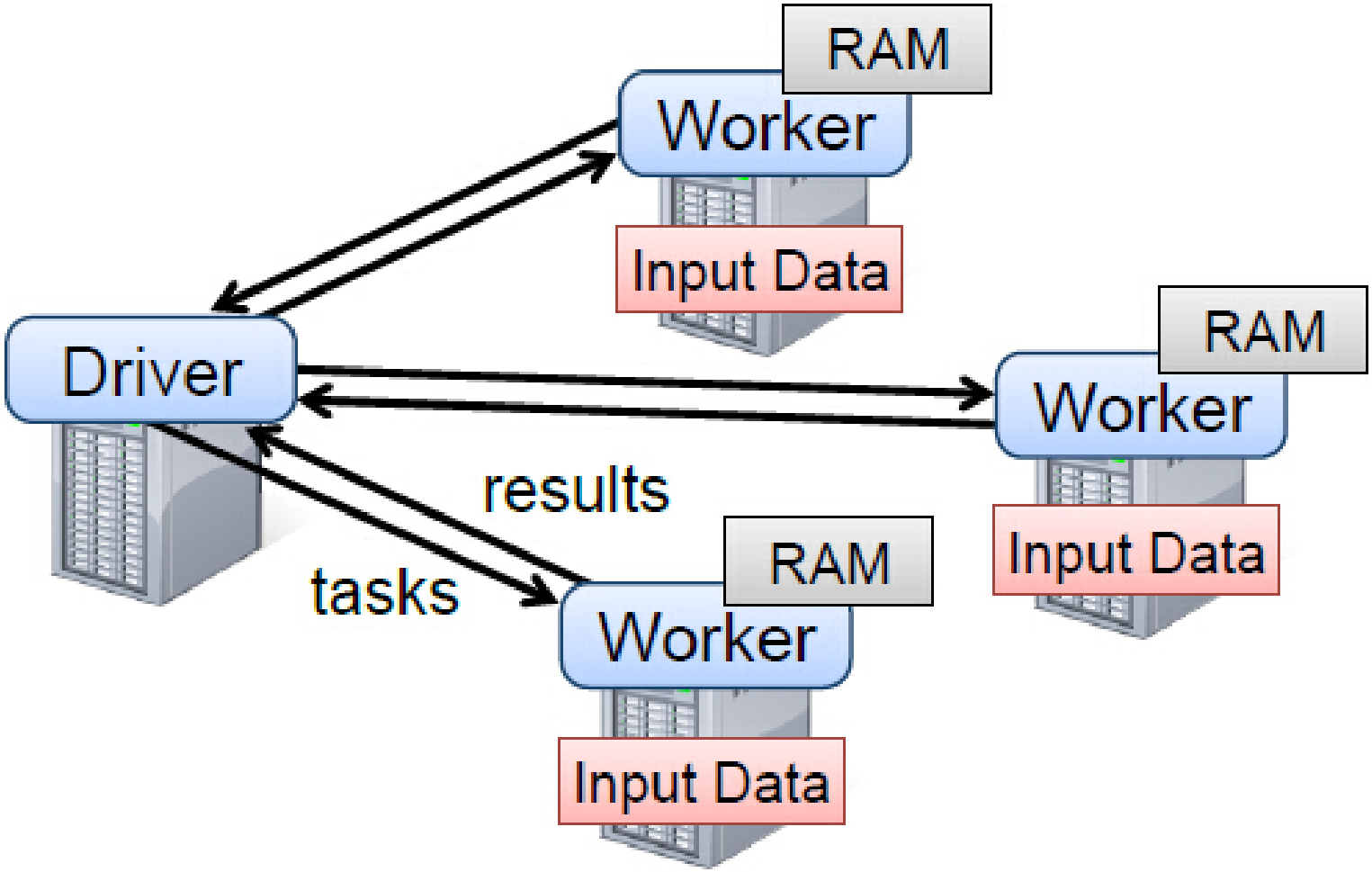
火龙果•整理
uml.org.cn

Spark目录

- Spark概述
- RDD
- 两个共享变量：广播变量、累加器
- 调度机制
- Spark性能实验

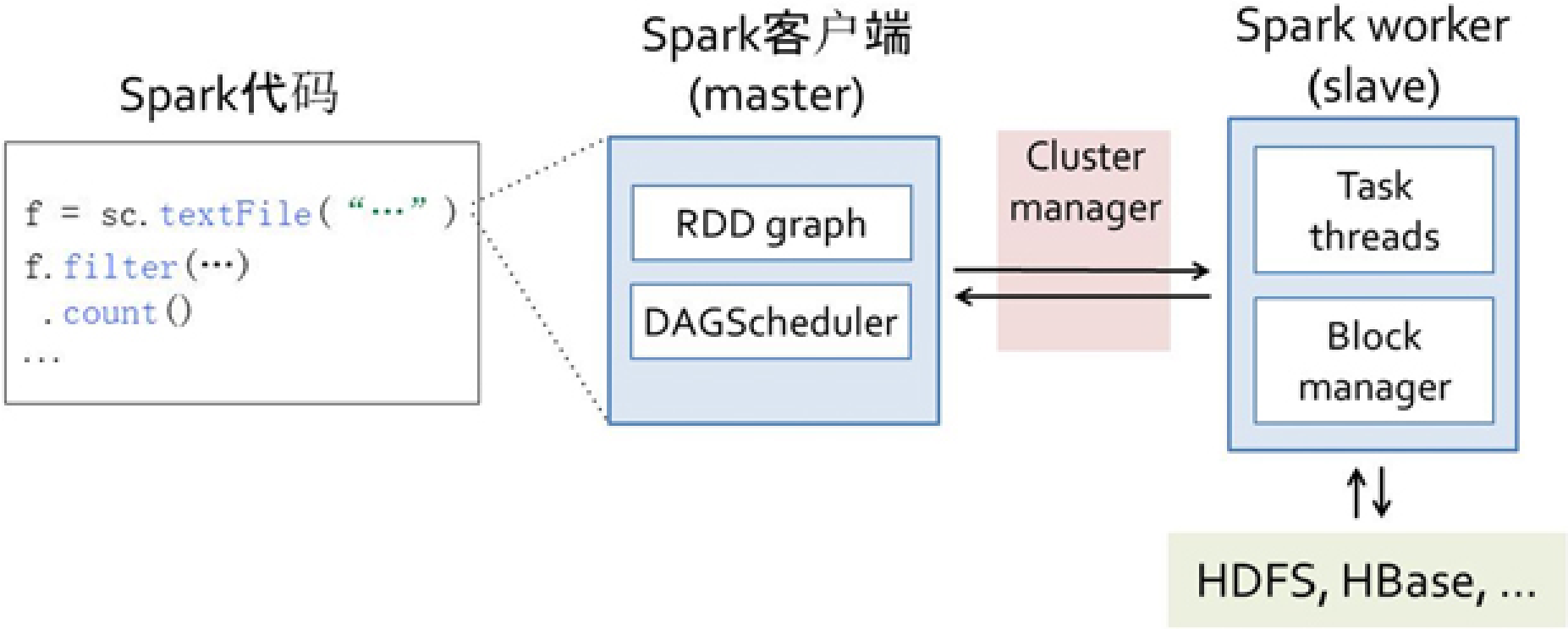
火龙果•整理
uml.org.cn

Spark Runtime



火龙果•整理
uml.org.cn

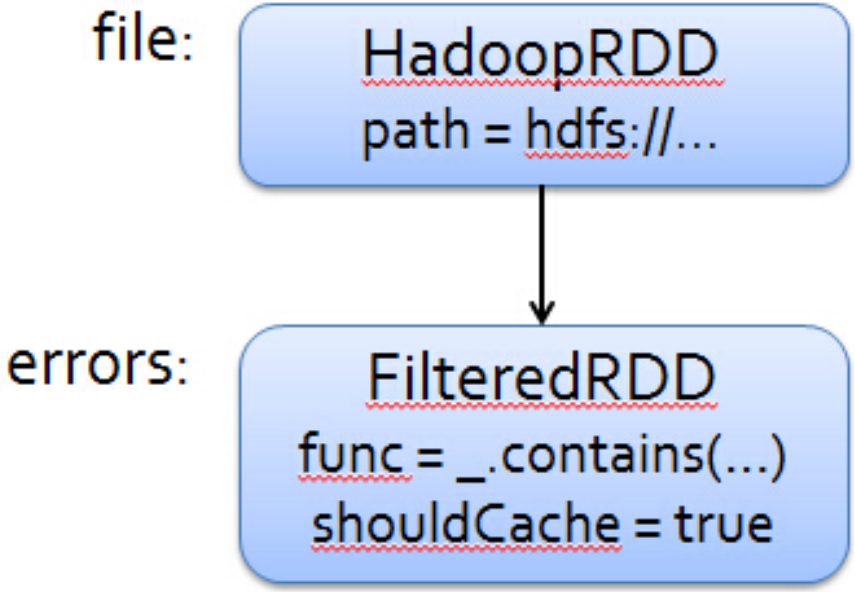
Spark程序的运行场景



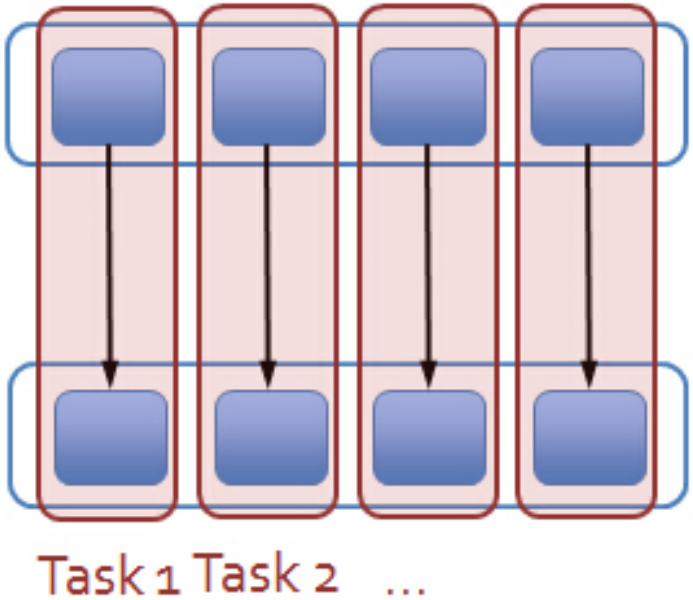
火龙果•整理
uml.org.cn

RDD Graph

Dataset-level view:



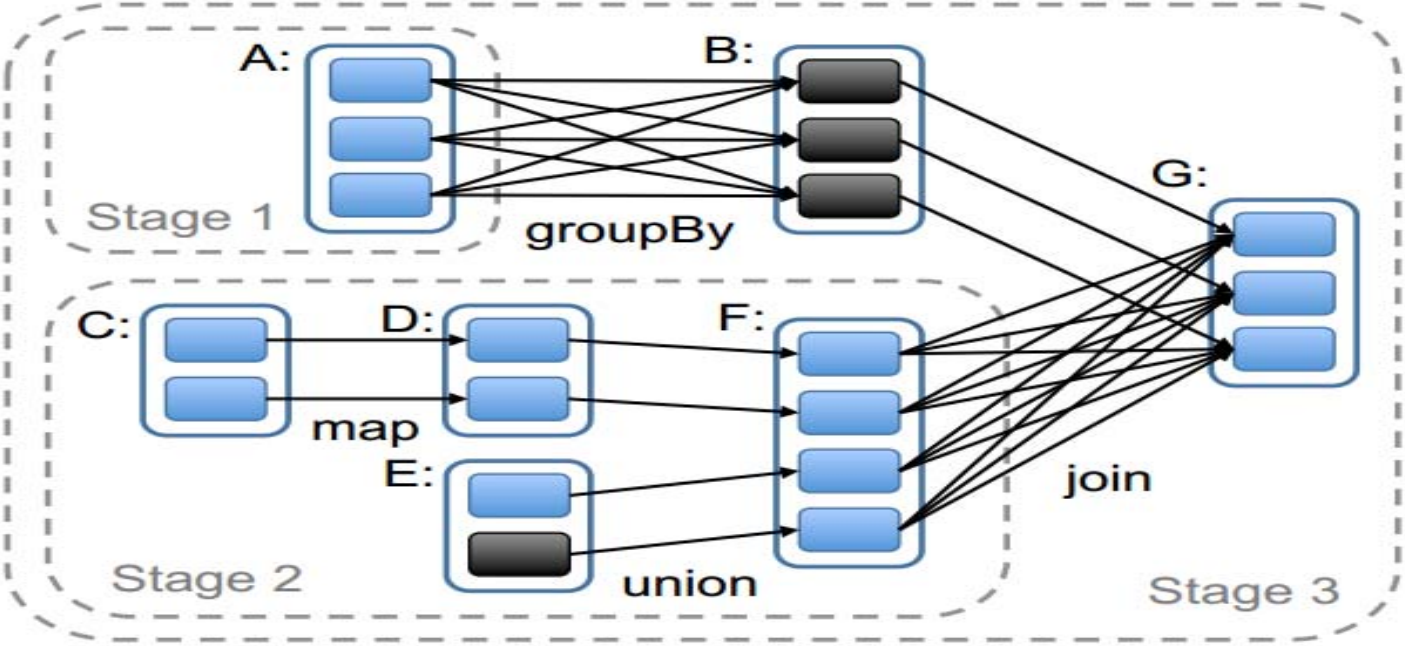
Partition-level view:



火龙果•整理
uml.org.cn

DAG

- DAG图由Stage组成
  - 每个stage里有一系列的task，task可以Pipeline的执行
  - Pipeline优化无需物化很多中间结果RDD，这将极大地提升性能
  - 每一个Stage中尽可能多的包含Narrow依赖的pipeline执行
  - Stage的边界是wide依赖所需要的shuffle操作或者已经计算好的分区



火龙果•整理
uml.org.cn

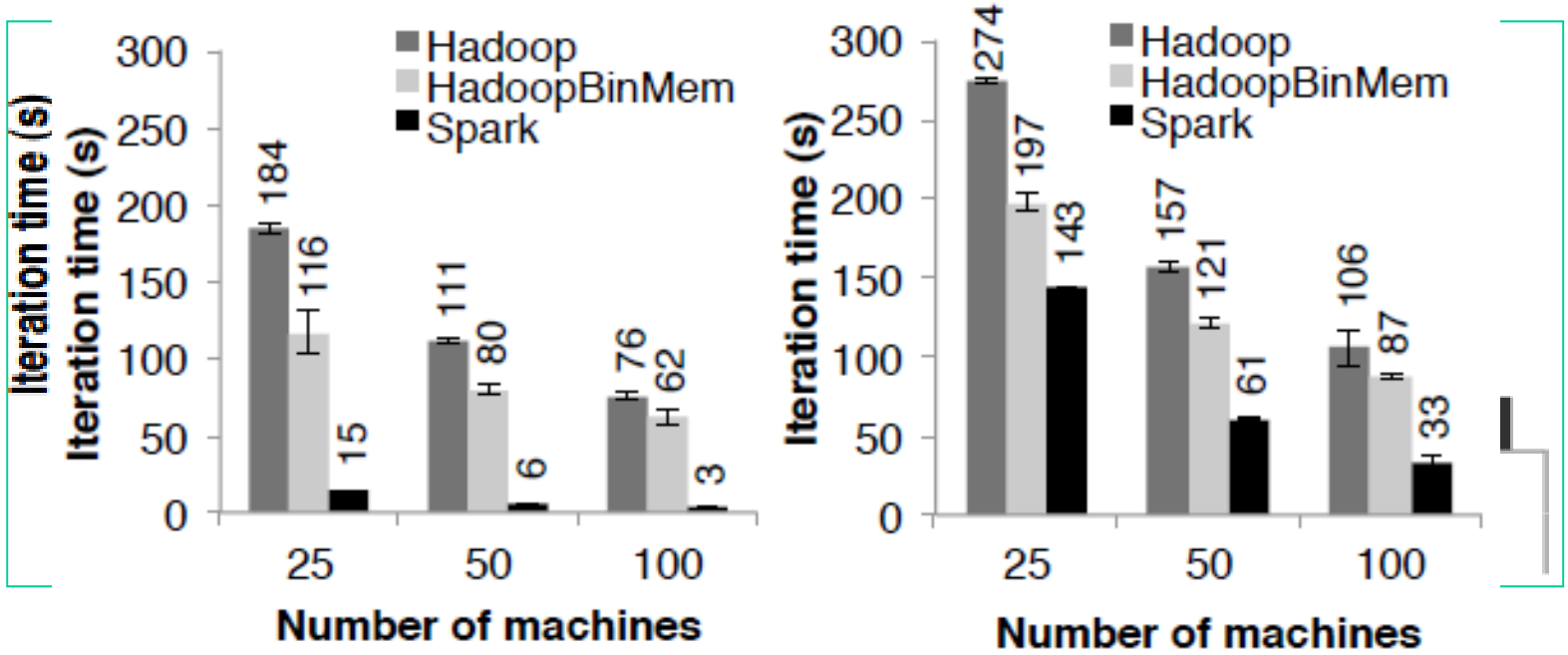
Delay Scheduling

- 基于数据所在的位置为集群中的机器分配任务，采用delay scheduling策略
  - 如果一个任务需要的数据分区在一个节点的内存中可用，就把这个任务分配给这个节点
  - 否则，如果一个任务需要的数据RDD提供更好的位置选择（比如一个HDFS文件），就把这个任务分配给这个节点
- 对于wide依赖，通过物化父亲分区的数据来实现简单的容错，类似于MapReduce的物化map端的输出

火龙果•整理
uml.org.cn

Spark性能

- Logistic Regression&K-Means算法
  - 100G数据100个节点的机器

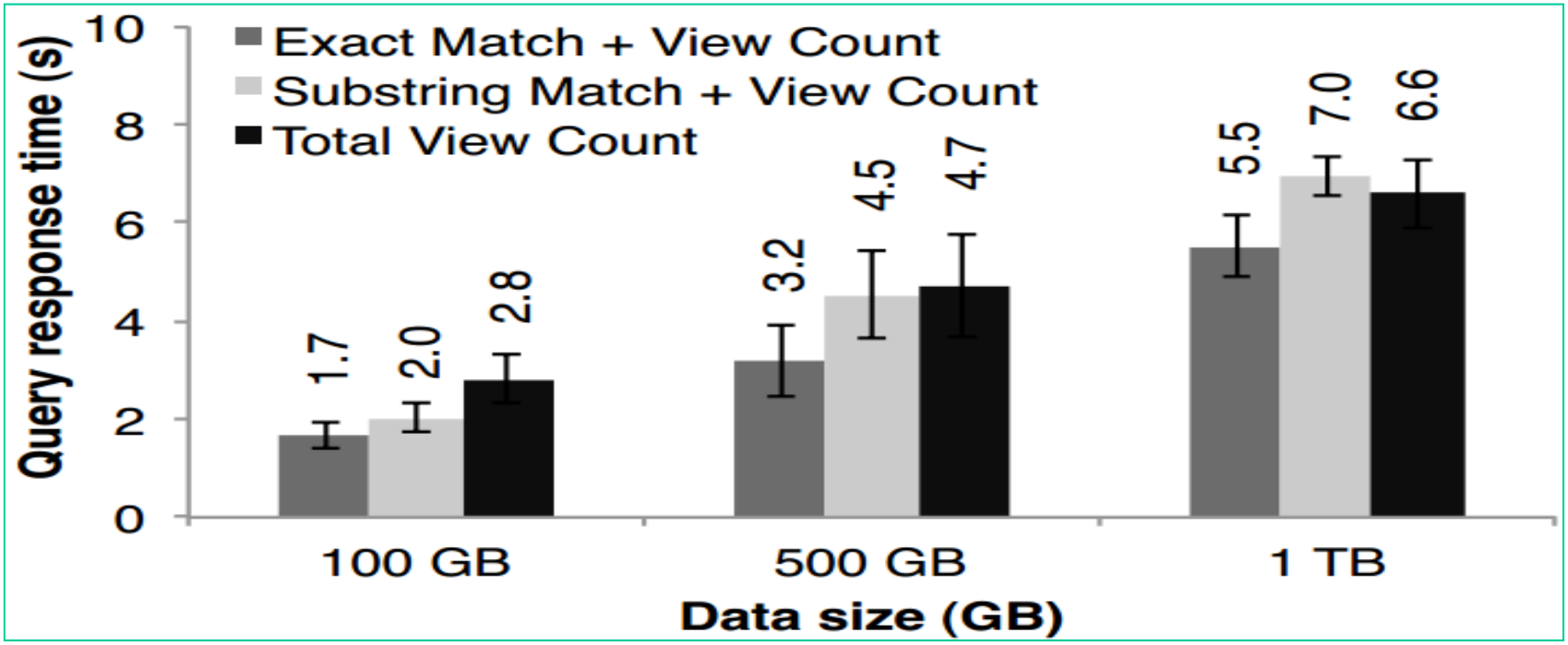


(a) Logistic Regression (b) K-Means

火龙果•整理
uml.org.cn

Spark性能

- 交互式查询
  - 1TB数据，100个机器（8核，68GB RAM），随着输入数据的增大，不同匹配程度下查询的响应时间
  - 在磁盘中查询1TB数据，要用170s



火龙果•整理
uml.org.cn

Part 2

SPARK STREAMING

火龙果•整理
uml.org.cn

大数据处理的类型

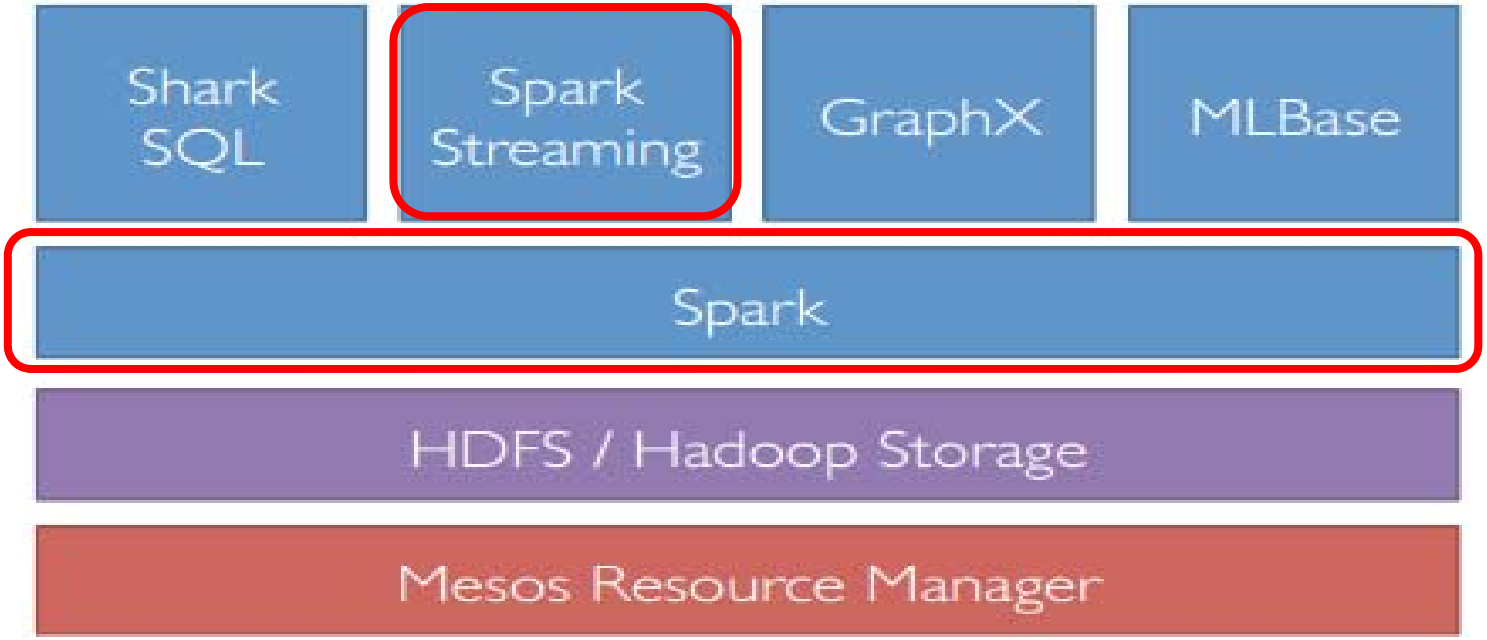
- 复杂的批量数据处理（batch data processing），通常的时间跨度在数十分钟到数小时之间。
- 基于历史数据的交互式查询（interactive query），通常的时间跨度在数十秒到数分钟之间。
- 基于实时数据流的数据处理（streaming data processing），通常的时间跨度在数百毫秒到数秒之间。

火龙果•整理
uml.org.cn

问题

- 三种情景的输入输出数据无法无缝共享，需要进行格式相互转换。
- 每一个开源软件都需要一个开发和维护团队，提高了成本。
- 在同一个集群中对各个系统协调资源分配比较困难。

- UC Berkeley AMP实验室数据分析栈



火龙果•整理
uml.org.cn

背景

- 大量应用需要实时流式处理
  - 社交网络，分析新的话题
  - 网站活动统计，测试用户访问模型
  - 入侵检测系统
  - ……
- 数据并行计算，需要分布式流处理模型

火龙果•整理
uml.org.cn

背景

- 要求：
  - 可扩展
  - 低延迟
  - 有效容错
  - 与批处理和交互处理结合
- 与批处理结合
  - 既保证低延迟的实时流处理
  - 又可以批处理TB级别的高延迟任务
  - 若不能整合，意味着需要两种编程模式和实现，需要双倍的工作量

火龙果•整理
uml.org.cn

Spark Streaming

- 大规模流式并行处理系统
- 思想：将流式处理当作短时间内的确定的批处理
- 解决前面所有问题：
  - 提供与批处理相似的高层API
  - 低延迟：使用RDD，中间状态保存在内存中，次秒级
  - 强一致性：所有数据都会执行确定的批处理操作
  - 有效的容错机制：与批处理一致
- 与Spark的批处理、交互处理，有一样的处理机制，一样的数据结构，一样的容错方式，因此可以很好的相结合
- 输入数据可以来自于：HDFS、Twitter、Flume、Kafka等实时数据

火龙果•整理
uml.org.cn

Spark Streaming

- 将流计算，当作一系列短时间内、确定的批作业
  - 将实时流每几秒切分为一个小的批处理
  - Spark将每个批处理的数据当作RDD，并使用RDD操作进行处理
  - 最后，将RDD操作的执行结果返回到批处理中
  - 批处理时间间隔小到0.5s

火龙果•整理
uml.org.cn

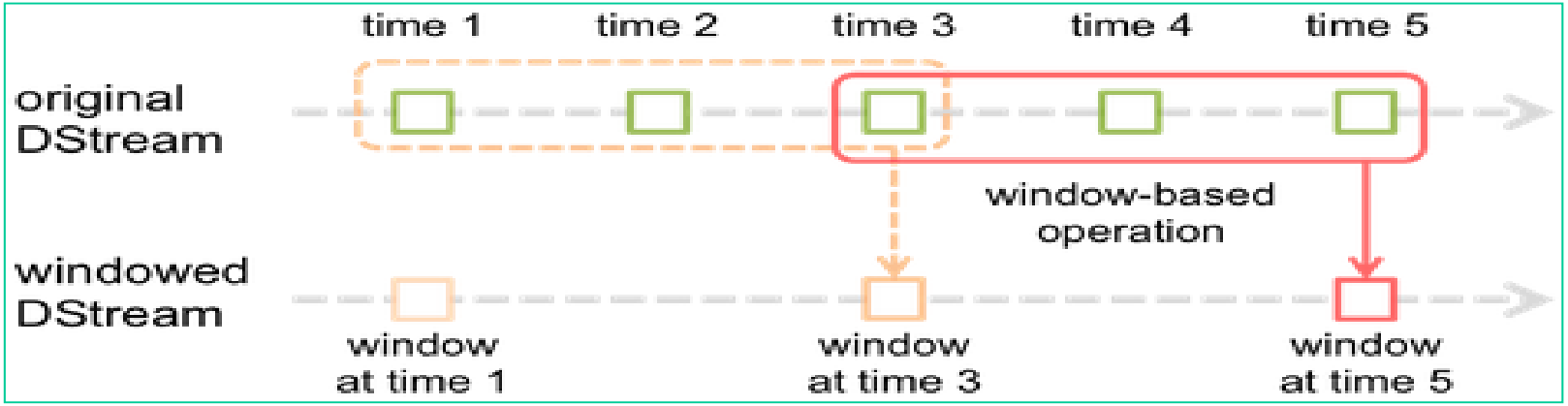
Discretized Stream

- Spark Streaming提供的一个编程模型，又称为DStream
- 可以由输入的数据流创建，或转换已有的DStream
- 用一系列RDD表示，每个RDD包含一定间隔的数据

火龙果•整理
uml.org.cn

Window Operations

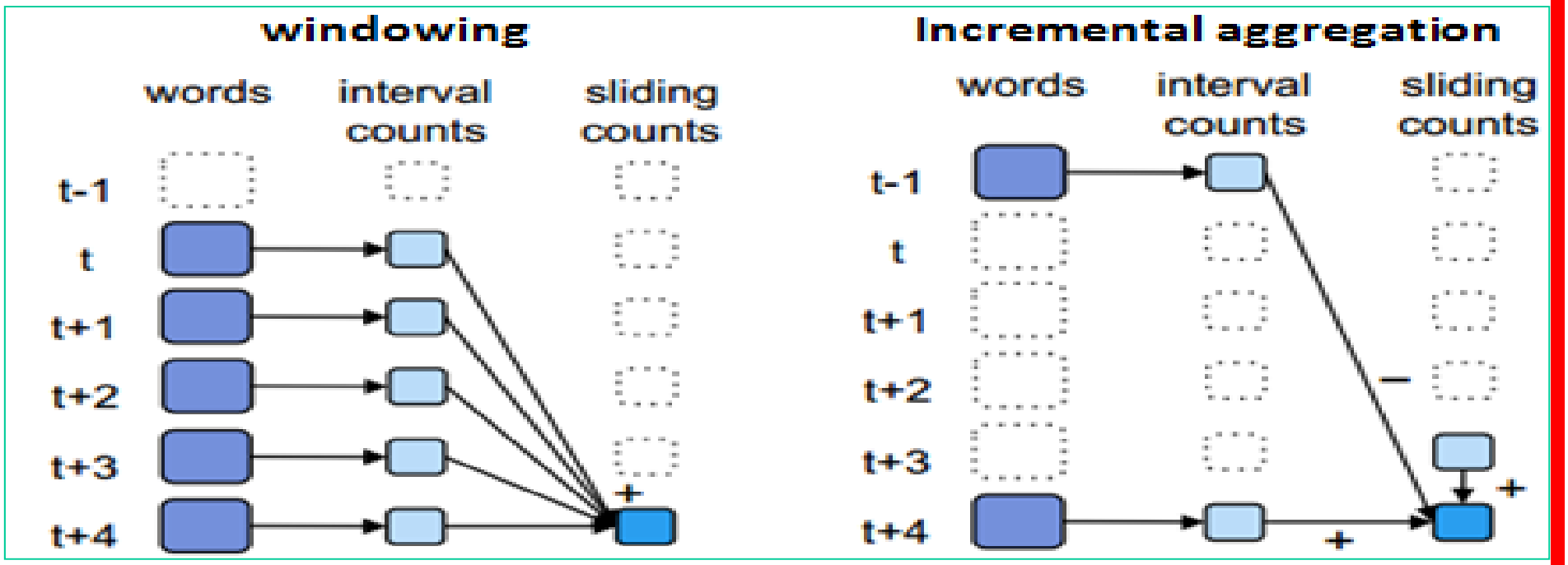
- Window操作对一个滑动窗口内的所有数据整合为一个RDD进行处理
- 参数：
  - window length：window包含的时间段
  - slide interval：基于window的操作执行的时间间隔，移动的步长



火龙果•整理
uml.org.cn

Incremental aggregation

- Incremental aggregation：增量式聚合



火龙果•整理
uml.org.cn

Performance Tuning

- 原则：
  - 1、有效利用集群资源，减小每个批处理的处理时间
  - 2、设置正确的批处理大小，使得数据处理跟上数据输入

火龙果•整理
uml.org.cn

减小批处理时间

- 并行的层次
  - 若并行的层次不够高，集群资源可能不被完全利用。
  - eg.分布式Reduce，默认并行任务是8，可通过并行参数的传递，或修改配置文件中默认并行数
- 数据序列化
  - Spark中序列化RDD数据。默认是序列化为字节数组，从而减小垃圾回收带来的停顿
  - 序列化输入数据。从网络中读取的数据是字节格式的，需要先反序列化再序列化为Spark的序列化格式。反序列化可能是瓶颈
- 任务发布的代价
  - 任务序列化：使用Kryo序列化，可减小任务的大小，从而减小发送到slave的时间
  - 执行模式：Standalone mode和corase-grained Mesos性能比fine-grained Mesos好

火龙果•整理
uml.org.cn

设置批处理大小

- Spark Streaming应用在一个集群上稳定，其数据处理能力必须跟上数据吸收的速度
- 正确的批处理大小，可以通过测试计算出
  - 用一个保守的批处理大小，和一个慢的数据输入流
  - 测试每个批处理端到端之间的延迟
    - 查看Spark log4j驱动的日志中total delay
    - 使用Streaming Listener接口监听
  - 若延迟与批处理大小可比的，则系统稳定；若延迟持续上升，则系统不能维持，不稳定。
  - 若有了一个稳定的配置，可以适当的增加数据输入速度或减小batch size
  - 注意：短暂的延迟增加可能是因为数据突增，只要延迟有降到一个低值即可（如：低于batch size）

火龙果•整理
uml.org.cn

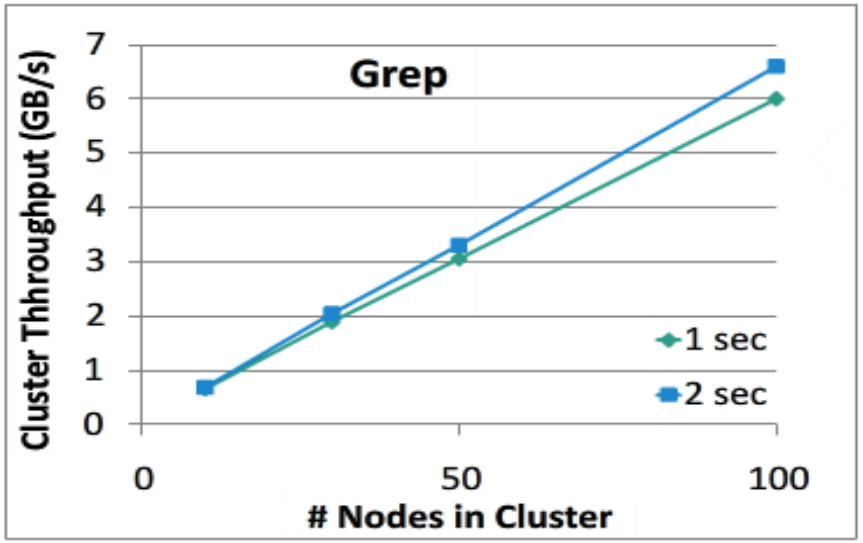
内存调优

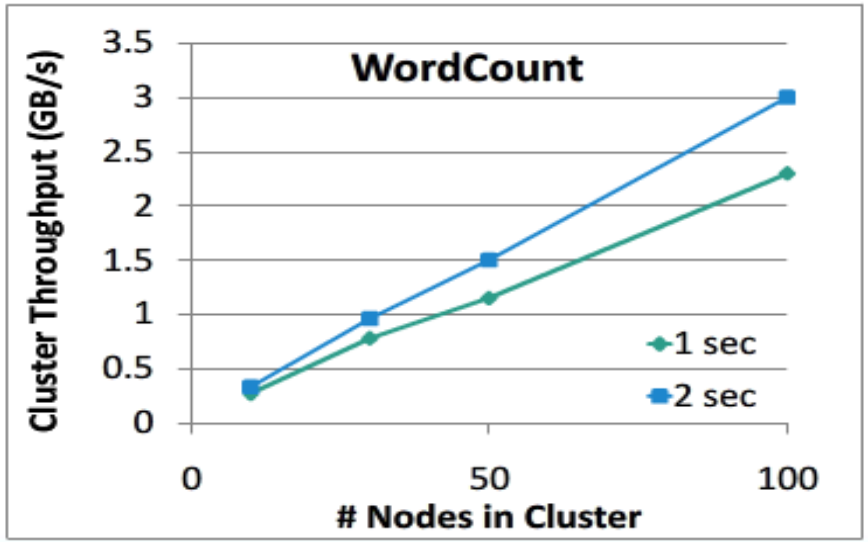
- 默认持久化级别：MEMORY_ONLY_SER，而在RDD中默认是MEMORY_ONLY。即使带来序列化和发序列化的开销，但是却大大减小了GC的停顿
- 清除持久化RDD：默认根据LRU从内存中清除持久化RDD。可以通过spark.cleaner.ttl的设置，使得大于某个时间的RDD被定期清除；或设置配置文件spark.streaming.unpersist为true，让系统判断哪些RDD不会再用，则不会被持久化。从而减小内存使用，改善GC
- 并发垃圾回收：使用并发的mark-and-sweep GC减小GC暂停的变化

火龙果•整理
uml.org.cn

性能

- 每条记录100byte，在100个节点上，速度达到6GB/s



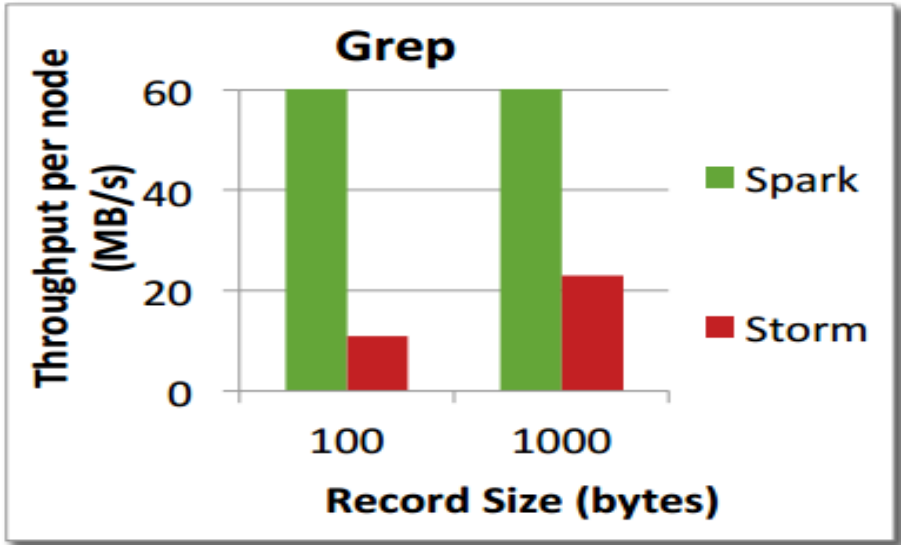


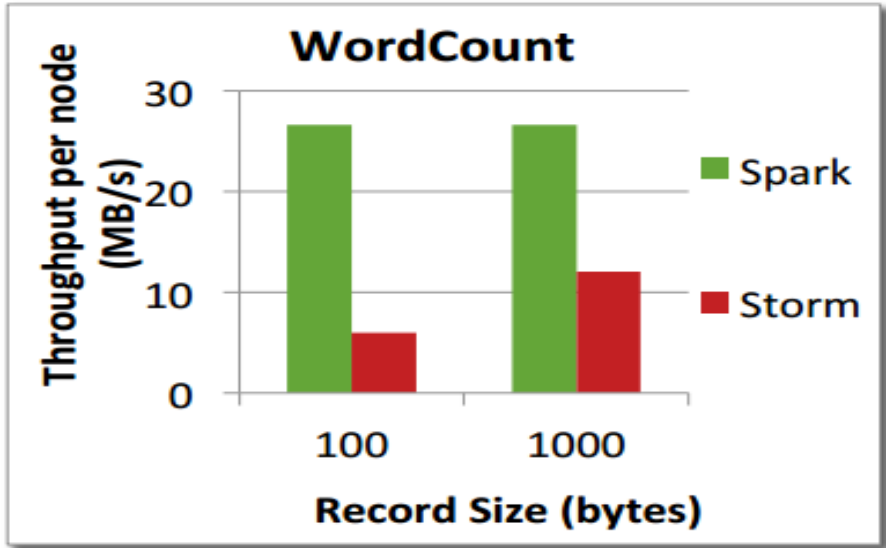
- 1s,2s：延迟——从流数据进入到返回结果的总时间

火龙果•整理
uml.org.cn

与其他系统比较

- 比Storm吞吐率高
  - Spark Streaming：670K records/s/node
  - Storm：115K records/s/node

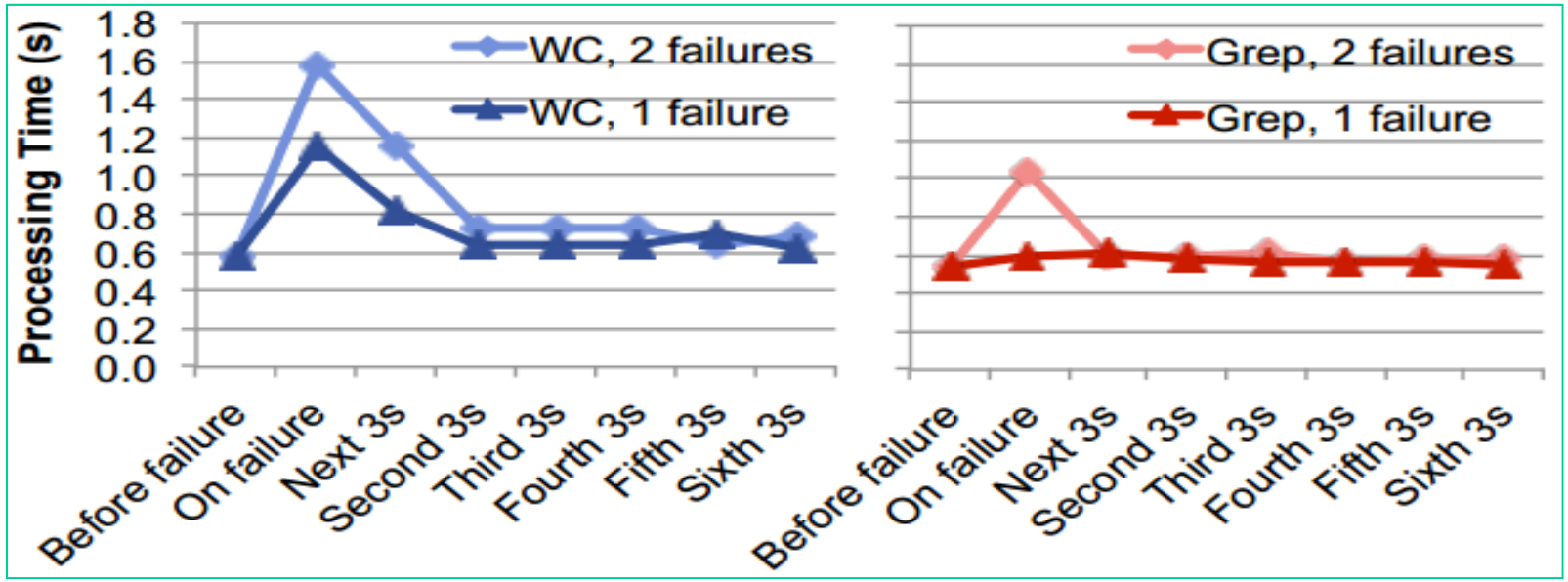




火龙果•整理
uml.org.cn

容错性

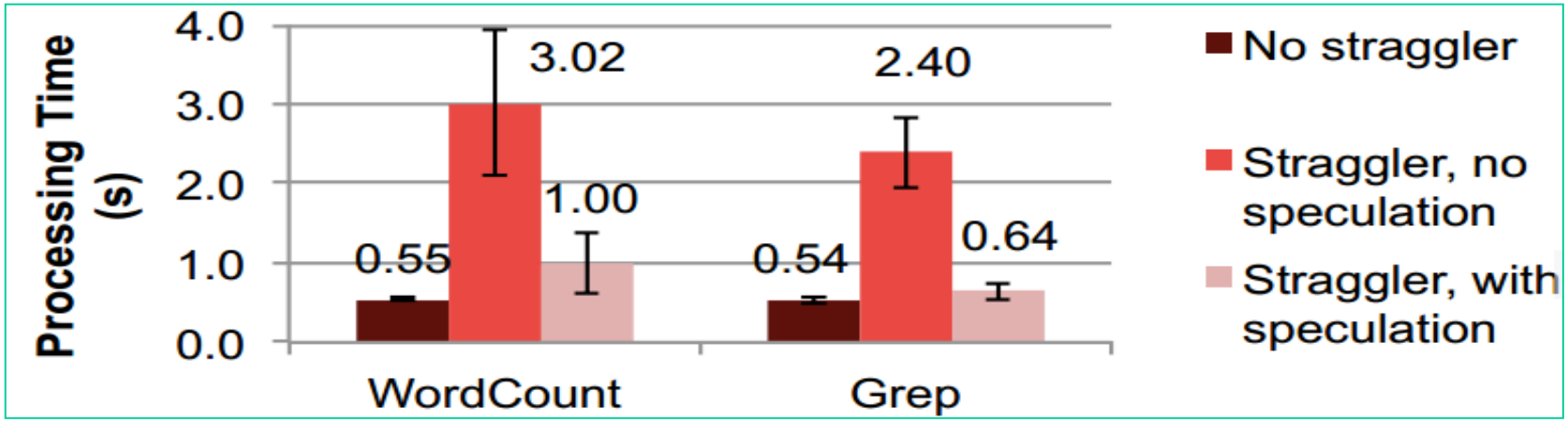
- WordCount&Grep
  - WC统计过去30s的值，有reduceByKey其lineage不确定，所以每10s一个checkpoint
  - WC失败后要重建之前的值，所有花费的时间稍长



火龙果•整理
uml.org.cn

Straggler

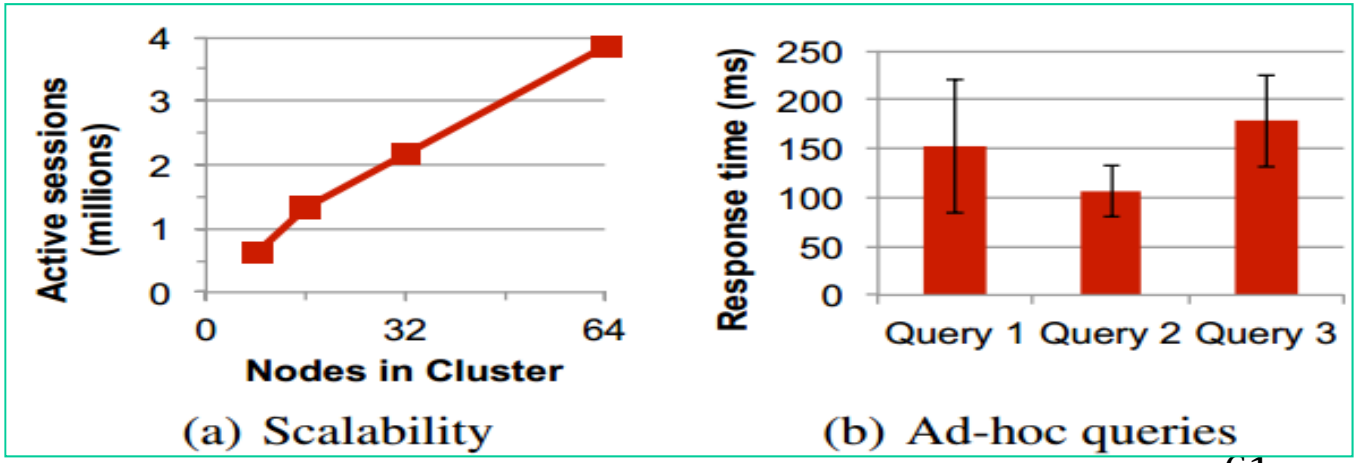
- 推测式执行大大改善了性能
- 实验中没有记住straggler，而实际使用时，会将straggler记入黑名单；说明若出现一个意外的straggler，也可以很快恢复



火龙果•整理
uml.org.cn

实际应用-Conviva

- 视频分布监控
  - a：扩展性，节点个数和对应会话数
  - b：三个专用查询的性能：所有活跃的会话数；指定用户的会话；失败过的会话



火龙果•整理
uml.org.cn

淘宝

- Spark on Yarn
  - 国内第一个基于Hadoop Yarn的Spark集群，基于云梯的5k台机器，单个作业可用内存500G，与HDFS无缝集成。用于数据挖掘推荐系统
- 淘宝的Spark之路
  - 2012年初Spark发布0.5，做第一次实验。在Mesos集群上用3台机器做实验，没有跑通生产级别的挖掘算法
  - 2012年6月中，Spark0.6版本推出Standalone模式。10台小集群，一周时间，跑通两个生产级别算法
  - 2013年上半年，云梯Yarn成熟；8月建立Spark on Yarn，最初只有100台机器，现在200台机器集群，可生产大批量的任务调度，可轻松的水平扩展