# 中国科学技术大学

# 研究生学位论文开题报告

论文题目 面向多核可伸缩的迭代式 MapReduce 的研究

学生姓名 俞玉芬

学生学号 SA14011084

指导教师 张 昱

所在院系 计算机科学与技术学院

学科专业 计算机软件与理论

研究方向 确定性并行

填表日期 2015 年 1 月 17 日

中国科学技术大学研究生院培养办公室

二零零四年五月制表

# 说　明

1. 抓好研究生学位论文开题报告工作是保证学位论文质量的一个重要环节。为加强对研究生培养的过程管理，规范研究生学位论文的开题报告，特印发此表。

2. 研究生一般应在课程学习结束之后的第一个学期内主动与导师协商，完成学位论文的开题报告。

3. 研究生需在学科点内报告，听取意见，进行论文开题论证。

4. 研究生论文开题论证通过后，在本表末签名后将此表交所在学院教学办公室备查。

# 一．选题依据

**１．阐述该选题的研究意义，分析该研究课题国内外研究的概况和发展趋势。**

**a) 研究背景：**

首先介绍所在课题组已经研发的一个确定性共享虚拟内存模型 SPMC，然后介绍 MapReduce 以及课题组之前基于 SPMC 实现的一个确定性 MapReduce 库，最后介绍迭代型计算以及传统的 MapReduce 处理迭代型计算存在的局限性。

**1) 确定性内存模型 SPMC**

随着多核机器的广泛普及，并行编程已成为有效利用多核机器的重要方式。由于多线程的交织运行，它们对共享区访问的次序是随机的，这将导致线程间交互的不确定性，这种不确定性给程序的调试、测试和重现 bug 带来了很大的挑战。现有的共享内存模型很难保证并行程序的确定性。为了解决这个问题，我们课题组提出在系统级强制确定性并行的单生产者-多消费者虚拟内存模型 **SPMC**，并基于该模型构建了一种确定性消息传递多线程编程模型 DetMP[1] (Deterministic Message Passing)。它向编程者提供了一组简单易用的编程接口，用于描述线程间确定的写-读共享关系，而由 DetMP 的实现保证线程间异步且确定的并发访问控制。

**2) MapReduce 及面向多核的确定性 MapReduce 库 DMR**

Google 公司提出的面向集群的 MapReduce[2]编程模型，极大的简化了并行编程。之后，Range 等人又提出了面向多核的 MapReduce 库 Phoenix[3]，并通过实验证明 Phoenix 可以获得相当好的性能。现有的面向多核环境的 MapReduce 库(如 Phoenix, Metis[4], TiledMR[5]等)都是基于传统共享内存实现的，其中线程间的交织运行会导致计算结果的不确定性。为此，课题组曾基于 SPMC 研制了确定的 MapReduce 库 **DMR[6]**。实验表明，相比 Phoenix，DMR 不仅可以保证 MapReduce 应用在多核机器上执行的确定性，而且对于非迭代式 MapReduce 应用，它表现出较好的性能和可伸缩性。然而，DMR 对于迭代型计算(如 Kmeans)则表现出相对较差的性能。

**3) 原有 MapReduce 库处理迭代型计算的局限性**

迭代型应用是一类非常重要的应用，现有的很多并行算法都是迭代型计算，特别是数据挖掘和机器学习中的算法，如典型的 Kmeans, PageRank[7], Hyperlink-Induced Topic Search(HITS)[8]等。它们的计算过程可简化为多次的 MapReduce 计算，即迭代式 MapReduce 计算。通常一次迭代的结果将被用于下次迭代计算，并根据两次迭代的结果判断是否需要继续下次的迭代。

目前大数据计算广泛采用的 MapReduce 库，如 Hadoop[9], Dryad[10]，它们都只关注于实现一次 MapReduce 过程。对于需要进行多次 MapReduce 的迭代型计算，这类传统的 MapReduce 模型的性能非常低下，分析其主要原因有以下几点：

- 开发者在使用传统的 MapReduce 进行迭代计算时，需要额外增加一个 driver 程序[11]，用于提交每次迭代的 MapReduce 任务，并在计算结束时负责检测终止条件是否满足。这个额外的 driver 程序会占用系统的 CPU，磁盘 I/O 等资源。
- 多次迭代间存在数据依赖，即下次迭代需要使用上次迭代中产生的数据。传统的 MapReduce 解决这一依赖的方式是：只在上次迭代完全结束后，才开始下次的迭代。这种迭代间的同步进行，将极大地限制计算的速度。
- 尽管有些数据在不同的迭代过程中不会改变，传统的 MapReduce 也会重复的加载。在分布式环境下，迭代之间通过网络传递这些数据，从而占用网络的带宽和磁盘 I/O，且带来不必要的延迟。

**4) 面向多核可伸缩的迭代式 MapReduce 库**

已有的 DMR 研究表明，它对非迭代型应用表现出较好的性能和可伸缩性，而对迭代型应用表现出较差的性能。分析该现象的主要原因，是由于 DMR 在每次迭代开始和结束时都需要分别创建和销毁确定性并行执行环境，包括子线程的创建和销毁。这种重复的创建和销毁为系统带来了很大的开销，从而降低了性能。

图 1 中展示了迭代型应用 Kmeans 的具体实验数据。其中 Init_time 表示创建线程的时间， MR_time 表示 MapReduce 过程的时间，Destroy_time 表示撤销线程的时间。从图中可以看出随着核数的增多，Destroy_time 明显增多，并且在 32 核下成为了 Kmeans 的主要的时间开销。

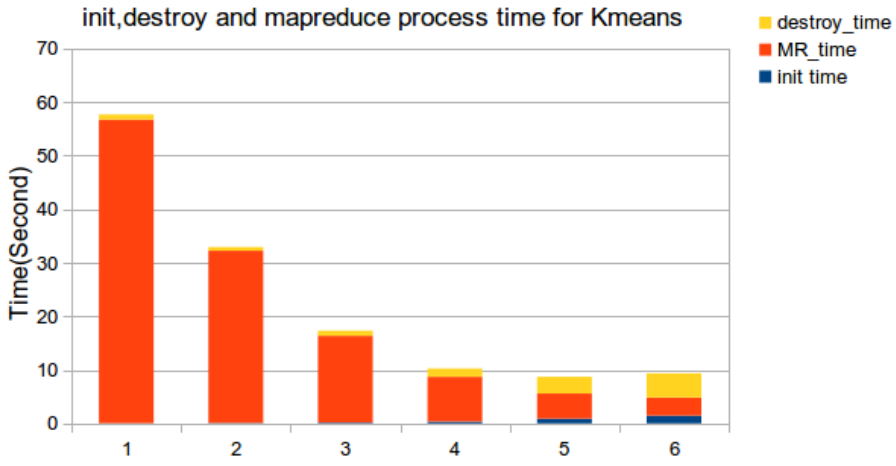


**图 1**：Kmeans 应用运行时各个部分的时间

为了有效支持迭代式 MapReduce 应用，本课题旨在研发一个面向多核可伸缩的迭代式 MapReduce 库。它将对原有 DMR 进行改进和扩展，并只在应用程序执行的开始和结束时才创建和销毁确定性并行执行环境，同时创建线程池，用于管理一组线程。当 map 或 reduce 任务来临时，线程池会将空闲线程与任务进行动态绑定；完成任务后，线程回到线程池中休眠，直到下次被唤醒。这种线程动态绑定的策略，可复用之前创建的子线程，而无需重复的创建和销毁，从而有效降低系统开销。

## b) 国内外研究概况：

近年来，为了使 MapReduce 模型更加高效的支持迭代式计算，学者们提出了多种迭代式 MapReduce 库，典型的有 Twister[12], HaLoop[13], iHadoop[14], iMapReduce[15]等。

### 1) 已有迭代式 MapReduce 库的分析

Twister 是一种基于流的 MapReduce 实现，专为迭代计算设计，Twister 允许缓存每次迭代的中间结果，即将 Reduce 的结果存到缓存中，供下次 Map 使用，这样不仅仅大大节约了运行时间，还节省了很多磁盘空间，Twister 每隔一段时间将 Map 任务和 Reduce 任务产生的结果写到磁盘上，当某个任务失败后，Twister 可以从最近的备份中，获得数据重新计算。缺点是 Twister 计算模型抽象度不够，支持的应用存在限制，无法使用户显示的控制数据划分和中间结果。

HaLoop 是对 Hadoop 框架的扩充，它将迭代型任务抽象为关系运算，HaLoop 将迭代计算中的静态数据缓存到本次磁盘，通过循环敏感的调度器保证前次迭代 Reduce 的输出和本次迭代 Map 的输入数据在同一台物理机器上，极大地减少了同次迭代过程中 map 和 reduce 任务需要被 shuffle 的数据量。缺点是 HaLoop 仅限于那些收敛准则依赖于最新两次迭代结果的算法，不具有带工作集应用的更通用抽象，只针对通讯模式受限的应用有效。

iHadoop 是对原有 MapReduce 框架的一种修改和优化，主要是通过修改数据流技术和任务调度，以进行高效的迭代式计算。在数据流方面，iHadoop 支持异步的迭代计算，即本次迭代完成之前，下次迭代计算已经开始，使迭代之间能够并行执行。任务调度方面，为了充分利用数据的局部性，iHadoop 调度器尽量将具有直接数据依赖的任务安排在同一个物理节点，从而减少网络带宽和磁盘 I/O 的开销。

iMapReduce 在 Hadoop 的基础上进行改进，它保持了 Hadoop 原有的编程接口，支持所有的 Hadoop 的应用，iMapReduce 将一对 map 和 reduce 任务进行绑定，并且只在初始化时，从文件系统中加载数据，经过多次迭代计算后，最终的结果由 reduce 上传到文件系统。每次迭代过程中，reduce 将得到的结果直接发送给与之绑定的 map，这样可以避免中间数据的重复加载和上传。同时 iMapReduce 支持异步迭代，即 reduce 可以将得到的部分结果提前发送给下次迭代的 map，map 一旦收到数据便开始工作，从而提升处理的效率。

2) **已有迭代式 MapReduce 模型的总结**

通过分析和总结现有的迭代式 MapReduce 研究，可以归纳出以下几点特征：

- 目前已有的迭代式 MapReduce 模型都是针对分布式环境。
- 在优化原有 MapReduce 库时，现有工作主要考虑的问题是：如何减少网络开销和磁盘 I/O，以降低迭代计算的开销。
- 为了加快计算的速度，有些模型(如 iHadoop，iMapReduce)采用了异步的方式进行迭代计算。

在分布式环境下，迭代间的数据通过网路传输，网络通信所占的开销将直接影响系统的性能。因此，之前关于迭代式 MapReduce 的研究集中于降低网络通信的开销和磁盘 I/O。事实上，分布式环境下的迭代计算，最终由某个物理节点完成，因此本课题结合已有的 DMR 研究，关注的主要问题是：在多核环境下，如何保证迭代型计算的高效性。

**c) 研究难点：**

由于为保证确定性，每个 DMR 线程实现为一个虚拟内存缺省为 COW(如堆、全局变量、栈等)的进程，进程间的写‐读共享只能通过显式建立的 SPMC 内存进行确定性通信。在这样的线程模型上实现迭代式 MapReduce 的难点在于：如何在保证确定性和可伸缩性的双重目标下，让执行下次迭代任务的各线程正确获得执行上次迭代任务的线程产生的结果以及主线程提供的下次迭代的初始设置，并有效地并发完成下次迭代任务。

２．国内外主要参考文献（列出作者、论文名称、期刊名称、出版年月）。


[1] Zhang, Y., Ford, B.: A virtual memory foundation for scalable deterministic parallelism. In: 2nd APSys (July 2011)

[2] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In 6th USENIX Conference on Opearting Systems Design and Implementation (OSDI), pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[3] Colby Ranger et al. Evaluating MapReduce for multi-core and multiprocessor systems. In 13th HPCA, pages 13–24, Washington, DC, USA, 2007.

[4] Mao Yandong et al. Optimizing mapreduce for multicore architectures. Computer Science and Artificial Intelligence Laboratory， Massachusetts Institute of Technology，Tech. Rep.2010.

[5] Chen Rong. Chen Haibo.Tiled-mapreduce: Efficient and flexible mapreduce processing on multicore with tiling. ACM Trans.Archit. Code Optim. 10, 1 (April), 3:1–3:30. 2013.

[6] Yu Zhang, Huifang Cao. DMR: A Deterministic MapReduce for Multicore Systems. International Journal of Parallel Programming, 2015, DOI: 10.1007/s10766-015-0390-5.

[7] Page L. The PageRank Citation Ranking: Bringing Order to the Web[C] Stanford InfoLab. 1999:1-14.

[8] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. J. ACM, 46(5):604–632, 1999.

[9] Hadoop. http://hadoop.apache.org/. Accessed July 7, 2010.

[10] Isard M, Budiu M, Yu Y, et al. Dryad: distributed data-parallel programs from sequential building blocks[M]. ACM, 2007.

[11] Mahout. http://lucene.apache.org/mahout/. Accessed July 7, 2010.

[12] Jaliya Ekanayake, Hui Li, Bingjing Zhang, et al. Twister: a runtime for iterative MapReduce. In 19th ACM International Symposium on High Performance Distributed Computing (HPDC), pages 810–818, New York, NY, USA, 2010. ACM.

[13] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. HaLoop: Efficient iterative data processing on large clusters. Proc. VLDB Endow., 3(1-2):285–296, September 2010.

[14] Eslam Elnikety, Tamer Elsayed, and Hany E Ramadan. iHadoop: asynchronous iterations for MapReduce. In 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pages 81–90, Washington, DC, USA, 2011. IEEE Computer Society.

[15] Zhang, Y., Gao, Q., Gao, L., Wang, C.: iMapReduce: a distributed computing framework for iterative computation. In: Proceedings of the 1st International Workshop on Data Intensive Computing in the Clouds(DataCloud ’ 11), p. 1112 (2011)

[16] Stephane Eranian, Perf wiki tutorial. 2013.

## 二．已取得的与论文研究内容相关的成果

**已发表或被接收发表的文章目录或其它相关研究成果。**

**1)** 已熟悉确定性的 MapReduce 库 DMR，能清晰理解它的设计思路和实现细节。

**2)** 确定了面向多核可伸缩的迭代式 MapReduce 的实现方案。

## 三．研究内容和研究方法

**主要研究内容及预期成果，拟采用的研究方法、技术路线、实验方案的可行性分析。**

**1、 研究内容和研究方法：**

本课题预期完成一个面向多核可伸缩的迭代式 MapReduce 库，它将提供一组简单易用的 API，应用程序可方便地使用这组 API 进行编程，以进行高效的迭代计算。该库具体的实现目标如下：

- 易用性和可伸缩性：基于传统共享内存模型实现的迭代式 MapReduce，为保证多个线程访问共享数据的一致性，往往需要进行同步控制。随着线程数目的增多，同步控制越复杂，开销也会越大。而基于 SPMC 实现迭代式 MapReduce，编程者只需简单的调用接口即可，无需考虑复杂的同步问题。
- 支持高效的流水：打破传统的 MapReduce 中 map 与 reduce 阶段之间的同步，实现流水。同时支持迭代间的异步进行，让计算更加高效。
- 支持较细粒度的 Fault Tolerance 和较好的调试功能：由于 SPMC 本身是个确定性的内存模型，所以基于它实现的迭代式 MapReduce 模型一方面支持较好调试功能，另一方面支持较细粒度的 Fault Tolerance。

为了实现该系统，需要研究的主要内容包括以下几点：

**1) 详细分析有关线程池的难点问题，并给出可能的解决方案**

首先，线程池的管理和任务的动态绑定。如图 2 所示，初始化时创建线程池，完成所有的迭代计算后销毁它。线程池应具备的功能有：当有 map 或 reduce 任务时，唤醒线程池中的空闲子线程(Worker)，将其与任务动态绑定；Worker 计算结束时，将本次计算的结果返回给主线程(Master)，然后进入休眠状态；Master 收集各个子线程的结果，更新相应的数据，判断是否需要继续下次迭代计算。

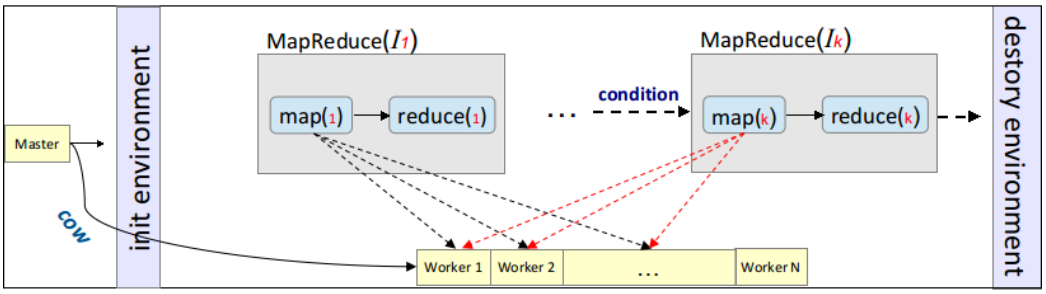


**图 2**：迭代式 MapReduce 的执行过程

其次，用于线程间通信的通道管理。DMR 中线程间的通信是基于 DetMP 建立的通道，该通道的优势在于用很小的虚地址段，传输无限量的数据。DMR 在初始化时，便会创建 map 线程用于完成 map 任务，reduce 线程用于完成 reduce 任务，同时创建它们之间通信的通道。迭代式 MapReduce 中，我们采用线程池和动态的任务绑定，因此线程间的通信是不可预知的，这将增加通道管理的难度，如何让线程池自动创建和使用通道是关键问题。

最后，主线程(Master)与多个子线程(Worker)之间的数据更新。每次迭代计算结束后，计算的结果以及全局的状态信息会被更新到 Master 中，它们会被下次迭代中的 Worker 使用，然而，由于地址空间的隔离，Master

地址空间中的新数据，无法为线程池中的 Worker 所知。因此，开始下次迭代之前，需更新 Worker 地址空间的某些数据，如何高效的更新这些数据将直接影响系统的整体性能。如图 3 所示，第 k 次 MapReduce 计算后，Master 和 Worker 相应的 Stack，Heap，Data 段上的数据会发生变化；为保持数据的一致性，第 k+1 次迭代之前，需要更新 Worker 的 Stack, Heap, Data 段上的相应数据。

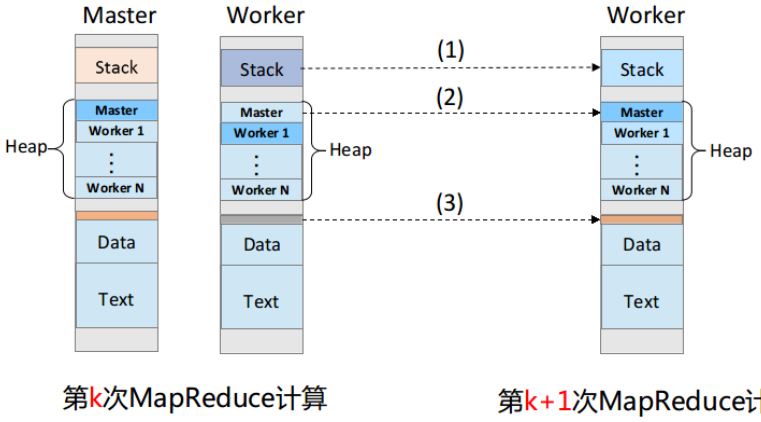


**图 3**：第 k 次迭代后 Master 与 Worker 地址空间的变化

**2) 迭代间异步算法的研制**

迭代间的计算有两种选择：同步算法(Synchronous)和异步算法(Asynchronous)，图 4 描述了同步和异步算法的数据流

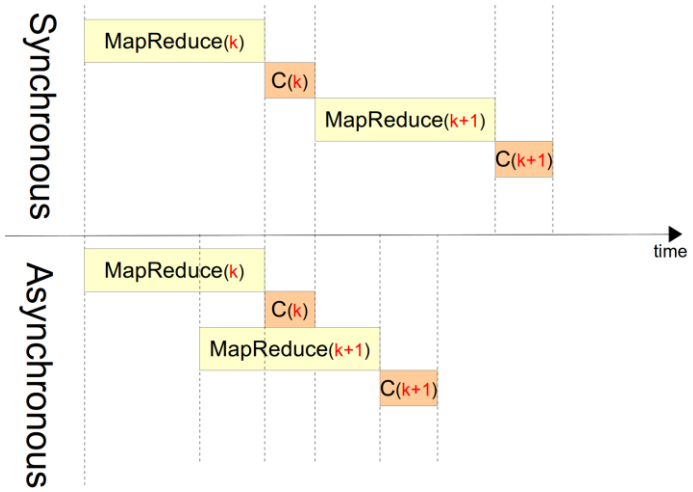


**图 4**：同步算法和异步算法的数据流

迭代间的同步算法（图 4 上方所示）：第 k 次的 MapReduce 计算结束后，进行终止条件检测(C(k))；如果条件不满足，进行第 k+1 次的 MapReduce 计算，即 k+1 次迭代必须等待 k 次迭代结束后才能开始。这种方式的优点在于：简单；缺点是：速度慢。传统的 MapReduce 模型在处理迭代型计算时，便采用这种方式。

迭代间的异步算法(图 4 下方所示)：第 k 次的 MapReduce 计算还未结束，已经开始了第 k+1 的 MapReduce 计算。这种方式的优点是：快速；缺点是：必须考虑迭代间的数据依赖关系。为了高效的进行迭代计算，本课题将采取异步的方式。

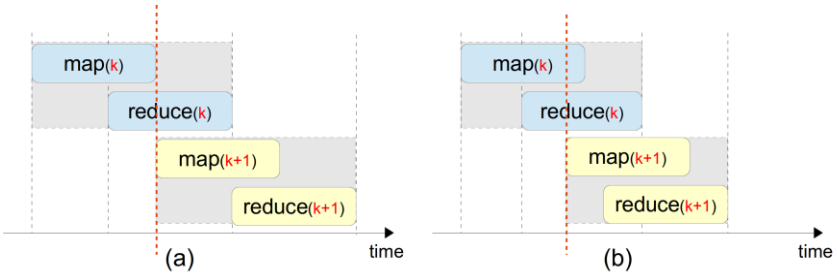


**图 5**:迭代间的异步算法的不同粒度

针对异步算法，有两种实现方案：

- 方案一：等到 map(k)完全结束，map(k+1)才开始计算。如图 5(a)。
- 方案二：不等 map(k)结束，而是只要 reduce(k)产生的数据可用，map(k+1)便开始计算。如图 5(b)。

这两种不同的方案，其实质是异步的并发粒度不同。如果选择粗粒度的并发(方案一)，优势在于数据依赖问题相对简单，但速度相对较慢。而选择细粒度的并发(方案二)时，速度较快，但是数据依赖问题复杂，系统处理该部分的开销也会增大。最终的实现中将采用哪种策略，需要结合具体的测试用例分析。

**3) 收集并改写典型的测试案例**

典型的测试案例应该极具代表性，应用广泛，同时能够很好的测试系统本身的性能。Phoenix 中已经给出了一个非常典型的迭代型应用——Kmeans。Kmeans 作为最为经典的基于划分的聚类方法，是十大经典数据挖掘算法之一。Kmeans 算法的基本思想是：以空间中 k 个点为中心进行聚类，对最靠近它们的对象归类。通过迭代的方法，逐次更新各聚类中心的值，直至得到最好的聚类结果。

为了有更多的测试案例评估本课题的研发成果，本人汇总了已有迭代式模型中使用的测试案例，其中较常使用的有 PageRank, SSSP(Single Source Shortest Path), Descendant Query 等。目前的测试用例都是使用特定编程模型的 API，无法适应多核环境下的 Phoenix 和 DMR，因此本课题需要改写这些测试案例。

**2、 采用的技术路线：**

**1) 路线**

- 调研已有的迭代式 MapReduce 库，分析它们的优点和缺点。
- 利用性能分析工具(如 Perf[16]等)分析 DMR 对于迭代型应用处理开销的各部分比例。
- 研发能兼容 SPMC 模型的新的线程池，以减少重复的创建和销毁线程带来的开销。
- 优化调度策略，使得在 key/value 数目较多时，流水线地执行各个阶段，以有效地提高的性能。
- 收集并改写测试用例。
- 编程实现基于 SPMC 的迭代式 MapReduce 库，并对其进行性能调优。

**2) 测试和性能评估**

- 使用典型的测试用例，将迭代式的 MapReduce、DMR 和 Phoenix 做性能的对比。
- 比较不同流水线粒度(流水的数据块大小)和不同流水线调度策略(流水的阶段数目)的性能。
- 分析程序性能存在的问题，以进一步的优化性能。

**3、 预期成果：**

- 实现迭代式 MapReduce 库，并期望其对迭代型应用的性能优于 DMR 和 Phoenix。

- 在核心期刊/会议上发表论文 1-2 篇。
- 完成硕士学位论文。

## 四．课题研究的创新之处

**研究内容、拟采用的研究方法、技术路线等方面有哪些创新之处。**

1) 基于 SPMC 实现线程池，并由线程池实现通道的自动管理。
2) 多次迭代之间实现异步执行，具备更加高效的流水。

## 五．研究工作进度安排

- 2015.07-2015.12 阅读分布式和多核环境下 MapReduce 的相关文献，熟悉理解已有的 MapReduce 库的实现和优化，调研已有迭代式 MapReduce 的研究，分析理解现有的各种迭代式 MapReduce 框架，完成迭代式 MapReduce 的实现方案设计。
- 2015.12-2016.08 根据迭代式 MapReduce 的实现方案，完成具体的代码实现。
- 2016.09-2016.12 测试迭代式 MapReduce 库的性能，撰写并投出论文。
- 2017.01-2017.05 进一步测试系统，完成硕士毕业论文。

研究生本人签名：_____

2016 年　　月　　日