分类号 TP391 学号 15063049

U D C 004.9 密级 公开

工学硕士学位论文

基于 Spark 的大规模图像库特征提取技术研究与实现

硕士生姓名 张新明

学 科 专 业 计算机科学与技术

研 究 方 向 内存计算

指 导 教 师 沈立　教授

国防科技大学研究生院

二〇一七 年 十二 月

Research and implementation of large scale image database feature extraction technology based on Spark

Candidate: ZHANG XinMing

Advisor: Prof. LI Shen

A dissertation

Submitted in partial fulfillment of the requirements

for the degree of Master of Engineering

in Information and Communication Engineering

Graduate School of National University of Defense Technology

Changsha, Hunan, P. R. China

December 6, 2017

独 创 性 声 明

本人声明所呈交的学位论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文题目：基于 Spark 的大规模图像库特征提取技术研究与实现

学位论文作者签名：张涛明　　日期：2017年11月14日

学位论文版权使用授权书

本人完全了解国防科技大学有关保留、使用学位论文的规定。本人授权国防科技大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：基于 Spark 的大规模图像库特征提取技术研究与实现

学位论文作者签名：张涛明　　日期：2017年11月14日

作者指导教师签名：　　日期：2017年11月14日

目　录

表 目 录

图 目 录

摘　要

当代社会互联网上图片的数据量急剧增长，而用户的检索需求越来越高，传统基于文本的图片检索技术因其描述词汇受限，很难在满足大数据背景下的图片检索。如何快速,准确的从众多图片中找到目标图片或者相似图片，成为了研究的热点。基于图像内容的检索技术在该背景下受到研究者们的关注，该技术使用图片自身的特征去匹配目标图片，检索结果具有非常高的准确性，百度和谷歌公司已经相继推出的以图搜图的服务，用户体验十分好。基于图片内容的检索技术包含了许多图像处理的技术，其中特征提取技术最为关键。

特征提取是基于内容的图像检索技术中的关键步骤，因为后续其他的图像处理步骤是在此基础上进行的。在众多的特征提取算法中，SIFT 算法最为著名，该算法提取的特征点与图像的旋转，大小无关，对于噪声，光线的容忍度也相当高，是一个划时代的特征提取算法。但是该算法的时间复杂度为指数级别，难以满足对处理时间要求特别高的的应用场景。因此后续有众多研究者对该算法进行优化研究，主要分为算法本身优化和通过硬件加速两方面。改进算法分别有 SURF、ORB 以及 MSERS 等。在硬件加速方式下，在 GPU 和 FPGA 下均有 SIFT 算法的实现。本文的工作和硬件加速是同一类型的研究工作，不同的地方在于本文是基于大数据处理框架 Spark 进行加速的，研究的是大数据背景下的特征提取加速。在众多大数据处理框架中，Spark 是一个内存计算类型的数据处理框架，在处理速度上有明显的优势。在此之前，暂时还没有研究者在 Spark 上进行大规模图像库特征提取工作的研究，于是本文基于 Spark 处理框架和 SIFT 算法，在上面开展大规模图像库特征提取的研究工作。

在本文中，我们设计了一个基于 Spark 的大规模图像特征提取系统 Spark-SIFT。该系统框架主要包含三部分：1）Spark 图像基础库 Spark-imageLib 模块；2）Spark-sift 特征提取功能模块；3）图片的序列化模块。之后本文又针对 Spark-SIFT 系统提出了三种优化方案。第一，因为图片的体积相对 Spark 来说普遍较小，而 Spark 在加载众多小文件时读写效率很低，针对这一现象，本文提出了 Key-Vaule 的图片描述方式，将图片转化成记录的形式，再将记录合并保存以提高 Spark 的加载效率。第二，Spark 在进行任务划分时仅考虑任务的总体积，而忽略任务中图片尺度大小，这一任务划分机制导致 Spark-SIFT 在处理图片大小相差较大的数据集时出现的负载不均衡问题，针对这一问题，本文提出了分割式特征提取算法，该算法核心思想是分而治之，先将大图片分割成小子块，并行处

理，之后再统一收集，通过这种方式避免因为处理图片的尺度大小而导致负载不均衡现象。第三，本文针对分割式算法中引入的 Shuffle 开销，进一步提出了 Shuffle-Efficient 特征提取算法，通过高效的分区策略减少跨分区收集同一张图片子块的网络开销。实验结果表明，本文提出并且设计的 Spark-SIFT 大规模图像特征提取框架取得了较好的加速效果。使用 7 台机器，处理 4G 图片集合，相对于单机提取，加速比达到了 19.5，优于 GPU 的加速比。Key-Value 的图片描述方式在加载 11G 图片数据集时，加载性能相对 binaryFile 方式提升了 61.7%, 相对于 objectFile 方式提升了 83.3%；分割式提取算法较不分割提取算法在处理 480M 图片集合将提取速度进一步提高了 7.8 倍；Shuffle-Efficient 特征提取算法有效的减少在收集图片子块时的网络传输开销，在处理 6.8G 图片数据集时，高效的分区策略相对于 Hash 分区策略，收集的性能提高了 29.7%。

关键词: **大数据**; **图像特征提取**; Spark; SIFT **算法**; **负载均衡**;

ABSTRACT

Nowadays, the amount of pictures on the Internet has increased dramatically, and how to quickly and accurately find the target pictures or similar pictures from many pictures has been become a hot spot of research. The content-based image retrieval has becomes more popular which has more accurate and better in user experience. The content-based image retrieval contains a lot of technique of image processing, including feature extraction technology which is most important technique.

Feature extraction is a key step in pattern recognition and content based image retrieval, because other subsequent image processing steps are carried out on this basis.SIFT algorithm is the most famous among other feature extraction algorithms and the extracted feature points is not effected by the rotation and size of image. What is more, its tolerance for noise, the light is quite high, So SIFT is a landmark feature extraction algorithm. However, the time complexity of the algorithm is exponential, so it is difficult to meet the requirements of real-time. Therefore, many researchers have optimized the algorithm, which can be divided into two aspects: the optimization of the algorithm itself and the acceleration by the hardware. The improved algorithms are SURF, ORB and MSERS respectively. Hardware acceleration is divided into GPU and FPGA. The work in this paper is similar as hardware acceleration, but it based on Spark ,a big data processing framework. In many large data processing frameworks, spark is a memory based data processing framework, which has obvious advantages in processing speed.Prior to this, yet no research on large-scale image database features in Spark extraction, this paper Spark processing framework based on the research work carried out in a large image database in the above feature extraction.

In this paper, we design a large scale image feature extraction framework based on spark. The framework consists of three parts: 1) image basic processing interface; 2) SIFT feature extraction algorithm; 3) image serialization. Aiming at the problem that the picture size is too large in the picture set, which leads to unbalanced load, we propose a segmentation feature extraction method. The large image is divided into small blocks, so as to improve the parallelism.In view of the shuffle problem in the segmentation algorithm, we further propose the shuffle-effient split type extraction algorithm.

Experimental results show that our framework achieves good speedup. With 7 machines, the 4G image set is processed, and the speedup is 19.5 compared with the single machine extraction. The split type extraction algorithm can further improve the extraction speed by 7.8 times compared with the non segmentation algorithm in dealing with the 480M image set. Shuffle-effient segmentation feature extraction algorithm effectively reduces the shuffle size, and further improves the feature extraction time.

Key Words: Bigdata; Image feature extraction; Spark; SIFT algorithm; Load balance

第一章　绪论

随着互联网的不断发展，以及终端设备，比如手机，拍照设备，监控设备的广泛应用，互联网上现在保存的图片数量巨大，并且每天都在以惊人的速度增长，有数据显示，2011 年 FaceBook 上就存储了多达 1000 亿张图片[1]。如何在巨大的图像数据集中精确并且快速的找到相应的图片，逐渐发展为一个十分热门的研究课题。该问题是大数据处理及图像处理两个计算机领域问题的交织。大数据处理技术在近年来无疑是最受关注的研究问题之一，其他的还有云计算及人工智能，三者有着紧密的联系。基于内容的图像检索技术 (content-based Image Retrivel 或者 CBIR)[2] 在大数据下的图像检索（Image Retrivel）[3] 背景下受到研究者们的重视，该技术充分利用图片自身的信息，不需要人工的为每张图片打上描述标签，较传统的以标签搜图技术，有极大的匹配精度。现在百度[4] 和 google 公司都有相应的以图搜图服务，用户通过上传相似图片，搜索系统将返回匹配成功的相似结果。CBIR 技术涵盖了许多图像处理中的重要环节，比如特征提取，特征匹配等，本文的研究工作是围绕特征提取进行的。

1.1　研究背景

本小节先介绍图像检索的基本概念，然后给出两种图像检索技术的实例，分别是基于文本的图像检索（TBIR）和基于图像内容检索（CBIR）。本文对两种技术进行比较，分析各自的优点和缺点，展现 CBIR 的存在意义及价值，最后对海量图像内容检索中的关键步骤展开分析。

1.1.1　图像检索

在 1970 时就出现了图像检索技术[3]，当时主要还是通过输入检索文字的方式来进行图片的搜索，比如 Getty AAT[5]，使用了 13300 个词汇来描述其下的所有图片，包括历史的，艺术的，建筑的等不同领域的图片。又如 Gograph，将其下的图片分为了 30 多级的检索主目录，每级目录下有含有子目录，层层递归，来划分图片的不同归类，以便于人们进行查找。上面两个例子就是以典型的文本的方式来进行图片检索的例子。

得益于研究者们的大量研究贡献，以文本的图片检索技术发展十分迅猛，现在已经有相当多的这种方式下的检索算法，比如 RageRank[6] 算法，聚类[7] 算法，语言解析算法等等，但是因为受到描述词汇的限制，描述词汇的模糊和多义性，在大数据背景下的图片检索，文本方式的图片检索效果并不理想。为了更加准确的表达搜索的意图，更加精确的搜索目标图片，基于内容的图像检索（Content-based Image Retrieval）[2] 技术逐渐被研究者们重视起来。

CBIR 技术和 TBIR 技术是两种不一样的检索技术，基于内容的图像检索技术利用图片自身的信息，这里的信息可以是图片的颜色，图片的形状或者图片的纹理，同时特征可以是全局的，也可以是局部的，该技术利用上述的这些特征，然

后根据相应的匹配算法去查找目标图像。因为查找的信息十分精确，没有描述模糊，描述歧义等限制，因此如果提取的特征的精度和匹配算法精度同时很高，那么 CBIR 技术的查询返回的结果非常准确。但是由于图片的特征信息量大，因此查找的速度也会受到其制约。目前，已经存在有基于内容图像检索的商用系统，比如 QBIC 系统[8]，WebSeek 系统[9]，PhotoBook 系统[10]，百度公司研发的百度图片系统及谷歌公司研发的 google picture 等。CBIR 系统的运行原理架构如图1.1所示:

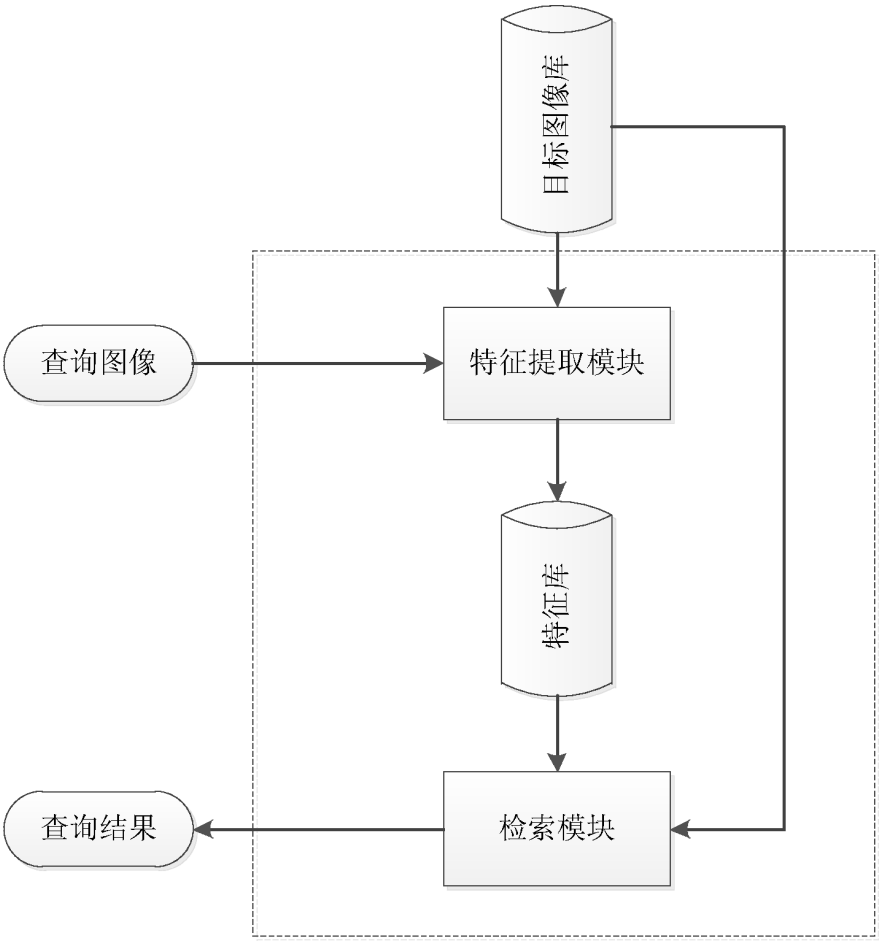

图 1.1　基于内容的图像检索框架

先将系统的图像库通过特征提取模块提取图片特征，然后所有被提取的特征将组成系统的特征库模块，查询图片先通过特征提取模块进行特征提取，然后将查询图片的特征和特征库中特征进行匹配，最后返回匹配的结果。

上面运行步骤中，特征提取，特征匹配是 CBIR 中十分重要的步骤，其中特征提取最为重要，因为如果特征提取做的不好，会直接影响后面的所有步骤，因此如何快速并且精确地提取高质量的图像特征，一直是研究的热点。

1.1.2 特征提取

因为在计算机中，我们输入处理的都是二进制数字信号，为了使计算机能够”理解“图像，我们必须从中抽取有用的信息，得到图像的”非图像“表示或描述，比如使用向量表示，从图像到数字符号的过程就是图像的特征提取。

1.1.2.1 特征分类

全局特征和局部特征是现在图像特征的大体分类，目前对于图像特征没有很明确的定义。常见的全局特征有图片颜色信息，纹理信息，轮廓形状信息，全局特征可以较好的理解图片的总体信息及内容，但是却不太适合进行图片的匹配工作，比如一张图片为红色，如果使用红色这一特征进行匹配，那么很可能搜到许多符合条件的图片。相反局部特征就很好的胜任图片匹配这一工作。常见的局部特征有，图像的斑点信息，图像的角点信息，图片的边缘信息及图片中的脊，局部特征是一张图片中具有局部不变性的，具有图片强烈代表性的局部信息，这些信息即使在图片受到光照，遮挡，旋转等外界影响的情况，也能很好的表示图像的独特性。因此局部特征较好的应用于图像的匹配工作，而不太适合从整体上理解图像，本文研究的 SIFT 算法就是一种局部特征的提取算法。以下是一些常见的全局及局部特征的定义解释：

- 颜色特征
  颜色特征[11, 12] 是对物体表面特性的一种描述，归类为全局特征。由于颜色特征没有很好的捕捉物体的局部特性，因此当在一个很大的数据库中，使用颜色特征进行图像匹配时，会检出很多满足要求的结果。颜色特征的优点在于其不受物体大小尺寸变化及物体空间位移变化的影响。

- 纹理特征
  纹理特征[13, 14] 是一种全局特征，和颜色特征相同的是，纹理特征也是描述物体表面的特性。纹理特征是对一个区域中的像素进行数学上的统计，根据统计的结果，代表图像的纹理信息。这种方式就和颜色特征提取的方式不一样，因为不是针对单个像素的。物体发生旋转时是不会影响物体的纹理特征提取，说明纹理特征可以对抗位置旋转噪声。纹理特征的缺点在于很容易受到分辨率及光照的影响，同时 3D 物体投射成 2D 时的纹理特征可能会发生变化。

- 形状特征
  各种基于形状特征[15] 的检索方法可以任意选取图像中的区域形状进行检索，简单快捷。但是该特征有几个缺陷，第一，该提取方法现在还没有完整的数学模型，因此是没有严格的数学意义上的等式证明，可能会存在不严谨。第二，当物体发生了一定形变时，基于形状特征进行匹配时就会失效，但是现实生活中物体的外部形状很可能发生变化，最简单的就是热胀冷缩。第三，3D 物体进行平面投射成 2D 时，由于视觉的变化，物体的轮廓形状是发生

变化的，因此对 2D 物体形状进行的特征提取得到的信息有可能和对同一个物体 3D 形态下的特征提取信息不一致，在一定程度上已经发生了失真。

- 空间关系特征
  空间关系特征是一种全局特征，它描述了图片中物体和物体间的空间关系，空间关系可以是邻接，可以是重叠，也可以是包含等，通过空间特征关系，我们就可以对图片中物体的整体位置关系有一个大体的位置感观。绝对的以及相对的空间关系是对物体间空间位置信息的两种归类。绝对空间关系描述的是物体间的绝对位置关系，通过绝对空间关系特征，我们可以描述出目标和目标间的距离大小以及方位。而相对空间关系特征描述的是物体间的相对位置信息，比如是 A 和 B 在空间中是上下关系，或者是 A 和 B 是左右关系等。相对和绝对空间关系是有一定关系的，比如我们可以通过两个物体的绝对关系推算出物体间的相对关系，但是反过来则不能。空间关系特征的优点在于可以很直观的理解整幅图片的物体的空间关系，但是空间关系特征相对简单，信息量小，对物体旋转，偏移，尺度变化等噪声抵抗力不好。如果图像仅用空间信关系特征进行配置时，往往是不能达到一个很好的效果，因为信息过于单调，所以可以用空间关系特征结合另外的一些图像特征，比如颜色特征及纹理特征，来进行目标物体的匹配。

- 角点
  角点[16, 17] 角点是一种局部特征，它通常位于图片中物体与背景交界的边缘处，具有高度的稳定性，可以代表物体的局部特性的一类点。

- 斑点
  斑点是这样的一个区域，该区域的色彩和四周有明显的灰度差异，如一个湖泊中的一只小船。斑点是针对一个区域进行提取的，提取的特征点的质量高于角点，能更好的代表物体的独特性，所以它在图像配准上扮演了很重要的角色。SIFT 算法中检测的特征点其实就是斑点，本文的研究也是在 SIFT 算法的基础上进行的。

1.1.2.2 提取算法

不同类型的特征对应着不同的提取算法，算法的原理不一样，提取精度不一样，计算复杂性也不一样。

- 颜色直方图法[18]
  通过将图片中的全局区域的 R,G,B 分量进行抽离，计算每一个分量占的比重，根据比重构造一个直方图，以上就是颜色直方图的基本原理。因为该方法是在物体的总体层次上进行统计和计算的，因此它和物体的空间关系是没有太大关系，当物体发生了旋转，尺度缩放等物理空间变化是，不是会影响颜色直方图法提取的结果。但是正是由于它对物理位置不敏感，会导致许多图片中物体关系完全不一致，但是它们计算出来的颜色直方图却是一样的，

这就是为什么有时候利用颜色直方图在很大的图片数据集下进行目标查找，查询结果包含许多无关的图片。

- 灰度梯度共生矩阵[19]
  灰度梯度共生矩阵（Gray-GradientCo-occurrence Matrix）方法是一种具有统计意义的纹理特征分析方法。该方法统计图像的梯度及灰度信息，然后使用一个矩阵保存提取到的信息。GGCM 的描述方法的优势在于可以很好的描述图像灰度和梯度的变化规律，同时也很好的描述了图像的纹理特征，特别适用于具有方向性的纹理描述。灰度梯度共生矩阵是一个计算效率很好的矩阵，因为在该矩阵的基础上，还可以统计出许多其他额外的信息，比如灰度及梯度的平均数值、灰度和梯度方差数值等等，有相关的研究者[20] 在研究灰度梯度共生矩阵的基础上，又提取额外的能量，惯量等四个维度的信息，精度相当高。

- 几何法
  几何法其实也是一种针对图像纹理信息的特征提取方法。根据纹理理论，纹理信息很多时候是具有规则的几何特征的，因此可以使用几何的方法来进行图像纹理信息的提取。分别是 Voronio 棋盘格特征法[21] 和结构法是几何法中很有权威性的算法，精度也十分高。

- 边界特征法
  该算法是一种针对图像的外围轮廓形状进行提取，获取物体形状信息。在该算法领域中，有许多有代表性的算法，在这里介绍两种比较常见而又经典的算法，它们分别是 Hough 变换检测平行直线方法[22] 和边界方向直方图方法[23] 。Hough 变换检测平行直线方法是将图片中物体边界处具有相同特征的点归类聚集在一起，最终将物体的外围形状轮廓提取出来。Hough 方法在物体空间参数不是很大的情况下，提取的总体效果还是很不错的。但是如果被检测的物体的体积很大，那么该算法计算的时间会很长，并且也需要开辟很大的存储空间来保存检测出来的边界特征点。边界方向直方图法其实也是利用直方图来获取物体的边界形状信息的，该方法首先会对物体边界求微分，在根据微分的结果做出一个直方图，最终获取物体的形状信息。

- 傅里叶形状描述符法
  傅里叶形状描述符 (Fourier shape deors)[24] 方法是对物体的外围轮廓进行 Fourier 变换数学运算，通过傅里叶变换函数的特性，物体的边界的连续性，边界性及封闭性均能很好的表现出来。同时傅里叶形状描述法还有降维的优点，可以将二维的物体空间转换为一维向量，从而降低了物体外围形状的时间复杂性。

- 尺度不变特征转换法
  尺度不变特征转换 (Scale-invariant feature transform 或 SIFT)[25] 是一种局部特

征提取算法，该算法先对目标图像构建尺度空间，这里的尺度空间在实际计算时是用高斯金字塔作为实例，随后算法就会在整组高斯塔上去搜索特征点，这些点不是普通的点，它们是可以十分准确的将物体的独特性描述出来的，即使物体被轻度的物理改变了，最后算法将这些点提取出来，使用 128 维向量来在数学层次上描述一个特征点，所有提出出来的特征点就形成了被检测物体的特征点集合。SIFT 算法提出来的特征点抗噪声能力十分强大，在物体被光照，遮挡，旋转等影响因素干扰下，算法依然可以十分高的精度将物体识别出来，是一个具有划时代意义的算法。

- Harris 角点检测法[26]
  该算法的基本思想是，使用一个的检测窗口进行在目标检测物体进行行列扫描检测，根据移动导致周边区域的灰度变化情况来判断这个检测区域是否存在检测。通常情况下，角点一般出现在检测区域和周边区域有着十分大的灰度落差的窗口上，当确定了区域窗口，在窗口上进行更细区域的搜索。检测的窗口一般是高斯函数，即使用高斯函数和图片的固定区域不断进行卷积操作。该算法检测出来的特征角点稳定性好，可以抵抗较大的物理噪声，同时算法的计算效率也较高。

在众多的提取算法中，综合考虑了算法提取的精度，算法的时间复杂度及空间复杂度以及可以突显大数据处理框架的在大规模处理时的高效性，本文选取了 SIFT 算法[27] 作为研究点，将其应用于大数据处理框架 Spark 上，完成大规模图像库的快速提取工作，本文将在第二章详细介绍 SIFT 算法的原理以 Spark 的基本原理。

1.1.3 特征匹配

除了上述讨论的特征提取技术之外，特征匹配也是图像处理中一个很关键的处理步骤，它们相互补充，如果特征点提取不好，那么匹配算法再好也没有用。反过来，如果匹配算法不够精确，提取的特征点再精确也不能实现精确查找物体。图像匹配问题实际就是相似搜索[28] 或者是近似搜索[29] 问题。相似搜索是指对于给定的检索的数据对象，根据预先设置好的相似阈值，在目标的数据集合中查找所有满足要求的目标对象，阈值设置的越大，返回结果的越多，反之越少。阈值就代表匹配相似度的评判准则，关键在于我们怎样去对相似度进行定义，什么维度的评判才是匹配时有价值的评判标准，常见的相似度的定义方式有下面几种：

- 汉明距离。给定比较的两个字符串 x 及 y，它们的汉明距离计算如公式1.1所示：

$$Ha = \sum_{1}^{d}(x_i = y_j) \tag{1.1}$$

- l_p 范数距离。给定 d 维实数空间 R_d 中的两个向量点 x=(x_1,...,x_d) 和 y=(y_1,...,y_d)，x 和 y 之间 l_p-范数距离的计算表达式如下:

$$L_p(x_i, y_i) = \left(\sum_{l=1}^{n}(x_i - y_i)^p\right)^{\frac{1}{p}} \tag{1.2}$$

- 余弦距离。给定 x 和 y，它们是 d 维实数空间 R_d 中两个非零向量，那么它们的余弦距离计算公式如下:

$$\cos(x, y) = \frac{xy}{|x|_2|y|_2} \tag{1.3}$$

- Jaccard 距离。给定两个集合 A 和 B，它们之间的 Jaccard 距离的计算公式如下:

$$Ja(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{1.4}$$

特征向量的维度越高，特征的描述也越详细，匹配的精度也就越高，如常见的 sift 特征是 128 维，Gist 特征是 960 维，深度学习提取的特征维度是 1024 维甚至更高。在面对着这些高维的特征向量时，如果完全单纯的计算每个向量间的距离，匹配速度将会十分缓慢，因此研究者们想追求的肯定是匹配的精度和匹配的，虽然很多时候我们要么割舍精度来换速度或者是降低速度来换取精度。面对高维特征向量，可以通过给特征建立有效的索引来提升查询效率，比如 KD-tree[30]，R-Tree[31] 或者 SR-tree[32] 等，这些方法的基本思想是将有相同特性或者是相近的特性的特征点统一划分到一个区域上，然后给这个区域建立一个索引，查找时根据索引先确定某个区域，然后再在区域中进一步查找，从而提升查找的效率。建立索引方式在面对着超高维特征向量时，效率还是偏低，有学者提出了降维的思想，将超高维向量降低到某个维度，牺牲部分精度来大幅度提升查询效率，这种思想的最著名实现就是局部敏感哈希算法[33]，该方法通过适当的调节参数，可以保证很高的精确度条件下也有很高的查询效率。

特征匹配方法在本设计工作中只是充当验证特征提取正确性的角色，并不是研究的重点，因此在第二章的基本原理部分没有对特征匹配进行深入的原理分析。

1.1.4 大数据处理处理技术

如今社会中的数字信息越来越多，各行各业现在都面临着大数据背景下数据处理的技术挑战，无论是处理的数据规模，还是处理数据的速度，单台机器早已无法满足当今大数据应用的处理要求了，因此急需一个大数据背景下的数据处理或者是存储框架，来帮助我们更加轻松的处理海量数据。自从谷歌公司提出了大数据的概念以及发表的三篇奠定大数据框架的论文之后，很多大数据处理框架在此基础下不断诞生。接下来，本文将会介绍内存计算引擎 Spark 和分布式文件系

统 HDFS 两个大数据处理框架，它们分别是本文设计工作中的计算引擎以及数据存储核心层。

1.1.4.1 内存计算框架 spark

Apache Spark 是专门为大规模数据处理而设计的快速通用的计算引擎，它的开发者为 UC Berkeley AMP lab (加州大学伯克利分校的 AMP 实验室)。Spark 的最大特点在于其内存持久化技术，可以将计算过程中的中间结果保存到内存，大大提升了计算任务的运算速率，十分适合迭代次数多的作业，比如像大规模图计算作业，神经网络作业等。Spark 本质还是 Mapreduce 框架，这点和 Hadoop 是相同，在 MapReduce 框架下，计算任务 Map Task 被分发到 Slave 上，计算结束后，通过 Reduce Task 统一收集 Master 上或者保存到 HDFS 中，区别还是在于 Spark 的中间结果支持内存缓存机制。Spark 提供了一套完整的内存编程模型及容错重算机制，使得开发者可以像编写普通程序一样去编写内存作业的应用程序，也不用担心缓存在内存的中间结果因为某些原因而丢失。

除了 Map 和 Reduce 操作之外，Spark 还提供了丰富的生态应用环境，比如提供了 SQL 与结构化数据处理工具 Spark SQL，机器学习工具 MLib，分布式图计算引擎 Graphx 以及流式处理工具 Spark Streaming。同时 Spark 的编程接口十分友好，支持 Scala，java,Python 和 R 语言。Spark 框架如图 1.2所示:

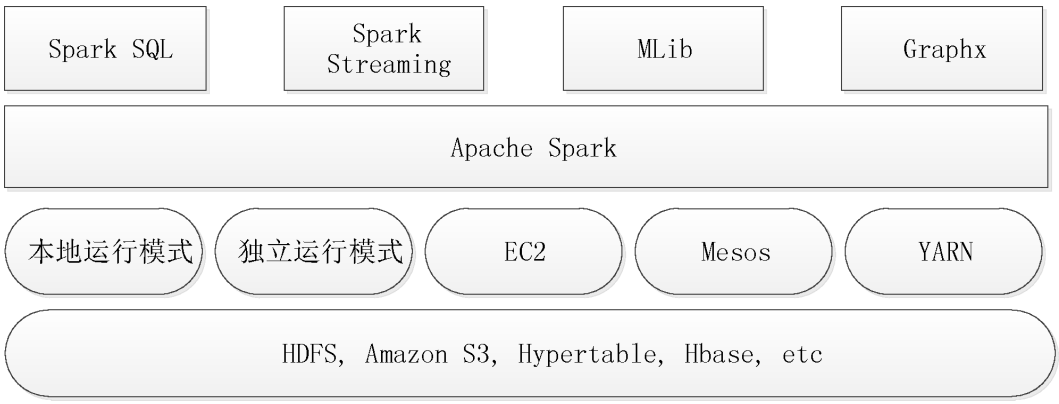

图 1.2 spark 框架

Spark 的核心是 RDD(Resilient Distributed Datasets)[34]，RDD 是一个分布式内存数据结构，可以将内存存储透明化，让用户显式地将数据存储到磁盘和内存中，并能控制数据的分区。RDD 的编程接口非常丰富，基本可以满足所有常见开发的需求。RDD 的接口操作也成为算子，比如 Map 算子，FlatMap 算子，filter 算子，这些算子都是 monad 模式的算子，即是不带 key 特性的单边数据处理。同时 Spark 的 RDD 接口中也提供了丰富的 Key 特性处理接口，比如 join，groupByKey，reduceByKey 等算子。通过这些运算算子，我们可以很容易的实现单机到分布式的功能转换。

在本文的设计工作中，我们将 Spark 作为大规模图像特征提取的计算引擎，加速特征提取的工作。

1.1.4.2 分布式文件系统 HDFS

在 BigData 背景下，需要被处理数据的规模往往是非常大的，很多时候达到了 TB 级别的，传统的单机的数据存储方式是无法满足读写速度要求及数据存储可靠性，在这种情况下，需要一种数据的分布式存储方案，将处理的数据分散存储而有集中管理，分布式文件系统 HDFS 就是这样的一种存储方案。虽然现在已有的网络文件系统（NFS）也算是一种分布式存储解决方案，但是相比之下，HDFS 存储方案的执行效率及安全性，是要远高于传统的的 NFS 架构。因为 NFS 本质上还是单机的存储方式，只不过访问端和存储端是处于分布式的状态，一旦客户端的访问量过大，就会造成服务端拥堵。而次，NFS 的安全可靠性也不如 HDFS，在 HDFS 的有完整的大数据场景下的文件备份方案，保证某个节点数据损坏后，可以迅速恢复。在数据的同步性上，HDFS 的效率会高于 NFS，因为在 NFS 下，客户端的写操作必须要先上传服务端，其他的客户端才能感知到数据的修改，在 HDFS 下则不需要先经过服务器端这一操作。

HDFS 是 Google 大数据三大论文中《The Google File System》[35] 的实现。整个 HDFS 集群是一个主节点和多个从节点的组成的系统架构，其中主节点称为 NameNode，从节点成为 DataNode。Namenode 复杂构建命名空间，存储 mete 数据，而 Datanode 则存储 data 数据。HDFS 整体框架图 1.3所示：

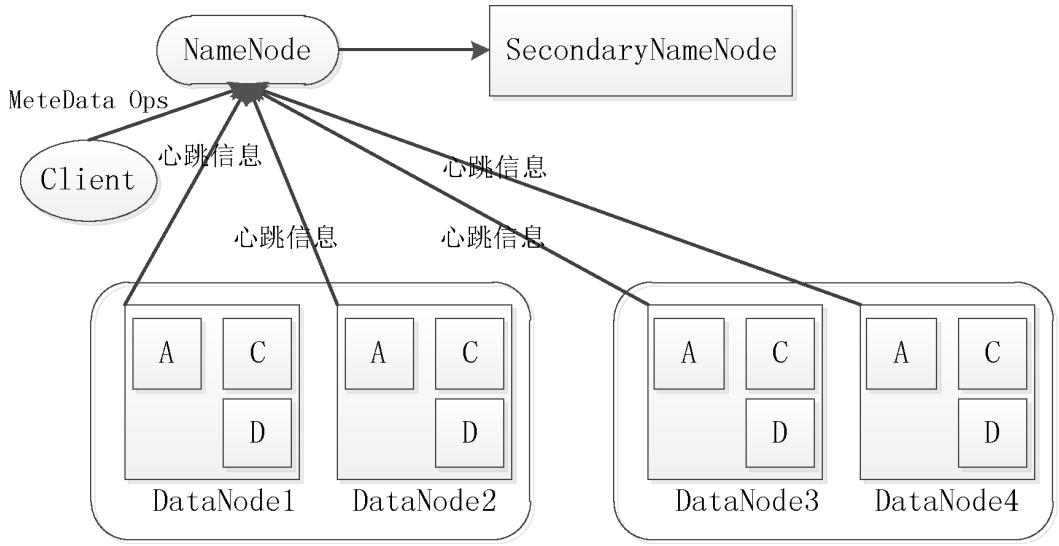

图 1.3 hdfs 框架

HDFS 在本文的设计中起分布式存储的作用，系统中的原始图像库，序列化图像库，特征库以及运行日志都保存在 HDFS 中。

1.2 研究问题及研究现状

本文研究的重点是如何加速大规模图像数据集的特征提取技术，因为根据之前分析，该技术在内容检索中最为重要，并且耗时十分长，因此该研究问题实际意义十分强。

关于图像特征提取问题的研究，很早就有学者开展研究了，而特征提取发展中具有里程碑式的工作应当是 Lowe 等人在 2000 年提出的 SIFT 算法，该算法具有局部不变的特性，意味着在物体被旋转或者是被遮挡的情况下，都不会影响图像的识别，并且算法对一些常见的物理噪声的容忍度也相当高。如果这些特征点进行匹配工作，即使是检索的图像数据集很大，也能很精确的查找出目标物体。但是由于 SIFT 算法时间复杂性高，特征提取时间难以满足实时要求，后续有大量学者在 SIFT 算法的基础上做优化工作，以提高特征提取的时间。

目前针对 SIFT 算法性能优化的工作主要体现在两个方面，一是在算法数学本质的基础上改进算法以降低时间开销，提高特征提取速度。例如 Bay 提出 SURF（Speeded Up Robust Features）[36] 延续 SIFT 算法的思想，通过积分图像和 Haar 小波相结合大大的提升了特征提取的速率；Ethan R. 等人提出的 ORB 算法[37] 采用了一种快速的基于 Brief 的二进制特征描述子方法提高了特征检测速度；Matas 等人则提出了最大极值区域（MSERS, Maximally stable Extremal Regions）特征检测方法[38]，MSERS 检测图像中灰度最稳定的区域，然后对检测区域进行旋转和尺度的归一。二是借助 FPGA、GPU 等硬件加速器提高 SIFT 算法的性能，例如 S.Heyman 等人将 SIFT 算法移植到 GPU 上[39]，获得了不错的性能加速。本文则研究如何使用 Spark 加速 SIFT 算法。

在大数据和图像处理的交叉研究中，也有许多出色的研究工作，例如，Hadong Zhu 等和 Akash K Sabarad 等分别在 Hadoop 平台下实现了大规模图像特征提取[40, 41]，相对于单机环境，获得了不错的加速比；Hanli Wang 等设计了一个云平台下的大规模图像检索的处理框架 CHCF[42]，Manimala Singha 等基于 Hadoop 设计并实现了一个基于内容检索的图像检索框架[43]。和上述工作不同的是，本文是基于 Spark 进行图像处理技术的研究。

大数据处理框架 Spark 自在 Apach 上开源以来，一直都受到热烈的关注，有许多研究者都使用 Spark 进行数据的处理，包括现在十分火热的机器学习，深度学习在 Spark 上都有相应的应用，比如基于 Spark 的机器学习库 MLib[44]，将 Spark 和 deep learning 结合的进行移动数据分析和研究[45]，基于 Spark 的对规模图计算[46]，但是我们发现 Spark 现在更多的是进行纯文本的分析，直接对图像进行处理的尚未发现，并且 Spark 里面也没有对图像处理支持的相关库，这促使我们在 Spark 上进行图像处理的相关研究以丰富 Spark 的生态圈，因此我们就选取了图像处理中的特征提取步骤，在 Spark 上开展研究。

1.3 本文工作与成果

如图 1.4所示，针对大规模图像特征提取，我们设计了一个基于 Spark 大规模图像特征提取系统 Spark-SIFT，然后，我们针对该系统提出了三种优化方案。首先，我们针对单个图片体积较少，而 HDFS 读写小文件效率又不高问题，我们对系统进行了序列化优化。然后，针对 Spark 现有的任务划分方式造成的负载不均衡问题，我们提出了分割式的特征提取算法。最后，我们在分割式算法的基础上，

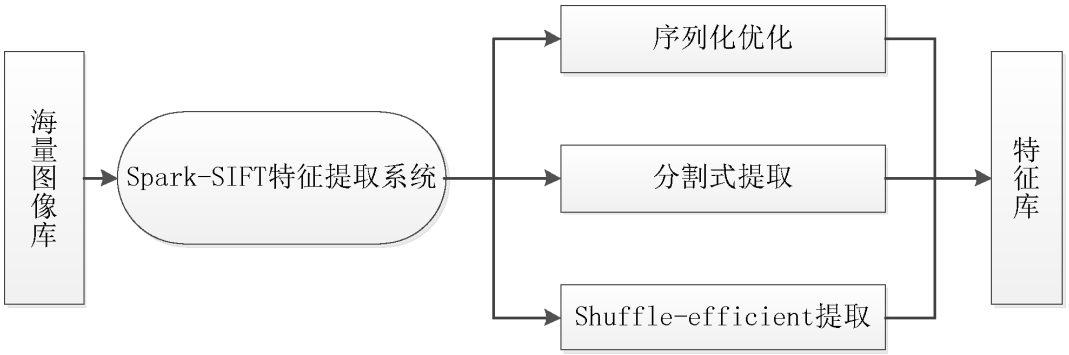

图 1.4　工作成果展示

进一步优化，提出了 Shuffle-efficient 的特征提取算法。详细研究内容可以归纳为以下 4 点：

1. 大规模图像库分布式提取系统 Spark-SIFT
   本文首先实现了一个 Spark 下的图像处理基础库 SparkImgLib，然后基于 SparkImgLib，本文实现了 Spark 框架下的 SIFT 算法，再在 Spark-sift 算法上的基础上，本文又实现了一个大规模的图像库特征提取系统。
2. key-value 图片描述式数据结构
   spark 在加载众多小文件时性能较低，而处理的图片体积一般是比较小的，比如几百 K，因此本文设计中采用用 key-value 的方式描述一张图片，将图片以 record 的形式序列化保存到 HDFS 中，从而使得 Spark 可以一次性加载多条记录，提升了读写的性能。
3. 分割式特征提取算法
   Spark 在进行任务划分时只考虑了 task 总体积，但是这种划分方式在本文设计中可能会造成 tasks 的负载不均衡，因为图片本身大小也会影响 tasks 的运行时间，针对这一点，我们提出了分割式的图片特征提取算法，将较大的图片分割成子块，来改善 Spark 在因为 tasks 划分方式导致的负载不均衡问题。
4. Shuffle-Efficient 特征提取算法
   Spark 中 Shuffle 机制是影响性能的重要因素。分割式提取算法中存在 Shuffle 问题，比如同一图片的子块散落在不同的分区时的数据混洗以及在不同分区上收集同一张图片子块时的数据混洗。针对上述 Shuffle 问题，我们提出了 Shuffle-Efficient 特征提取算法，提升了 Shuffle 操作的性能。

1.4　论文结构

全文分为 6 个章节，总体结构如图 1.5 所示：

第一章为绪论，在绪论中，我们首先讨论了 TBIR 技术和 CBIR 技术的区别，分析了两者的优缺点及使用场景，通过对别突显了 CBIR 技术存在意义，进而引出了 CBIR 中的两个关键技术，分别是图像特征提取技术和图像匹配技术。对于特征提取部分，本文首先对常见的特征进行分类，解释每一种的定义及区别，接着又针对不同类别的特征给出相应的特征提取方法。对于特征匹配部分，我们解

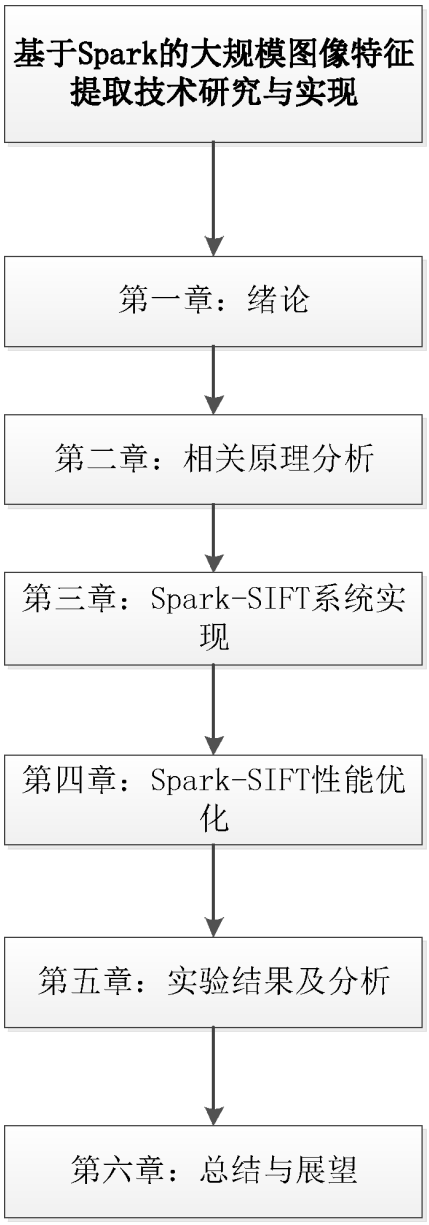

图 1.5　论文组织结构

释了匹配的基本原理及几种度量方式的特点和区别。在介绍完 CBIR 关键处理技术后，我们又介绍了本文设计工作中所涉及到到的大数据处理技术，在这部分内容中，我们主要介绍了本系统设计中的两个大数据处理框架 Spark 和 HDFS，我们分析了它们的存在的价值，技术特点，及在本文设计中扮演的角色。最后，介绍了本文的研究问题及问题研究现状，本文工作成果以及论文结构三部分。

第二章本文分析了设计中所涉及的一些原理知识，它们包括图像处理中的数字图片表示原理，特征提取算法 SIFT 算法原理，大数据处理框架 Spark 及 HDFS 分布式文件系统原理，以及 GPU 特征提取原理，这些原理知识为本文的设计工作提供理论基础。

第三章介绍了大规模图像库特征提取系统 Spark-SIFT 系统的整体设计工作，先介绍整个 Spark-SIFT 系统的整体框架，然后针对框架中的每个模块，包括序 Spark 图像处理基础库 Spark-ImageLib 模块，序列化预处理模块，分布式特征提取功能 Spark-sift 模块，分布式存储模块以及 Spark-SIFT 系统容错模块等进行详细的分析。

第四章介绍了在 Spark-SIFT 基础上提出的三种优化方案，分别是 Key-Value 的图片描述方式，分割式特征提取算法以及 Shuffle-Efficient 特征提取算法。

第五章进行性能数据的分析，分别针对 Spark-SIFT 系统的三种优化策略性能以及 Spark-SIFT 系统的综合性能进行对别分析

第六章总结了本文的所有设计工作及创新点，分析了存在的不足，最后对未来工作的一些看法和展望。

第二章　相关原理分析

在本章中，将针对本文涉及的关键技术的原理进行分析和解读，这些关键技术是指导本文工作开展的理论支撑，只有把这些技术原理充分理解后，才能解决设计过程中遇到的难题，才能进行一系列的优化工作。这些技术原理包括，图像表示原理，SIFT 算法原理，spark 内存技术原理，spark 任务调度原理，spark 性能优化原理以及 HDFS 存储原理。

2.1　图像表示原理

在计算机中，图像的数字表达形式其实是一个二维矩阵，矩阵中的每个数值就是我们常说的图像的像素点。在现实生活中，我们看到的图像大多数是彩色的，在这种情况下，图像数字化表达时，一个像素点值就包含了三个分量，分别是 R,G,B，这个三个分量的取值为 0 到 255，三个分量的不同取值和混搭，就构成了我们现实生活中看到的所有色彩。但是在图像处理中，灰度图像更为常见，因为除了色彩信息之外，灰度图像中包含了所有彩色图片包含的信息，不会有任何的区别，包括像形状特征，纹理特征等等，但是灰度图像需要的保存空间比彩色图像的空间更小，因为在灰度图像表示中，像素点只包含一个分量，所以大大缩小了保存的空间以及提升了读写的速率。本文的设计工作就是建立在灰度图像之上，我们将原始的图片都进行了灰度化处理，后续的所有处理都在此基础上进行。

灰度图像的每个像素用一个字节保存，从彩色图像转变为灰度图像，可以通过对彩色图像的三个分量进行加权平均来得到灰度值，加权公式如 2.1所示：

$$Gray = 0.11B + 0.59G + 0.3R \tag{2.1}$$

理解好基础的图像原理，是 Spark 下图像基础库 SparkImgLib 实现的理论指导，因为 SparkImgLib 包含了在 Spark 下图像的表示以及一些基础处理接口。

2.2　SIFT 算法原理

在本小节中，将会对 SIFT 算法进行详细分析，它是 Spark-SIFT 系统中最核心的算法。

在如上一章的介绍，SIFT 算法一个提取局部特征的算法，它通过构建物体尺度空间，在尺度空间中搜索满足要求的特征点，这些特征点具有很大抵抗物理噪声能力，比如旋转噪声，光照噪声，遮挡噪声等，这些点在 Lower 论文中被称为具有尺度不变性的特征点。它的基本特点如下：

- 局部不变性好，在物体受到比较严重的物理噪声干扰下，依然可以准确的描述出物体原有特性。
- 特征信息丰富，一个特征点是采用长度为 128 的一维向量进行表示。

- 可扩展性强，很容易和其他形式的特征向量结合在一起，使得特征信息更加丰富。

SIFT 算法大体可以划分为 5 个步骤，分别是高斯塔建立，高斯差分塔建立，极点检测，消除边缘响应，关键点方向的分配和描述，接下来，将会对这几个步骤进行分析：

1. 高斯塔建立

   SIFT 算法中的尺度空间的具体表示就是图像的高斯金字塔（gaussian pyramid），从2.1图就可以看出来，为什么图像的尺度空间被称为 gaussian pyramid，同一张图片的不同尺寸，从小到大，依次往下排列，形成了塔状。建立 gaussian pyramid 时，先将一张进行高斯模糊及尺寸放大，得到其以下面一层的图片，然后再在新的图片的基础上继续做高斯模糊及尺寸放大。具体的层数是算法的一个参数，SIFT 论文中一组金字塔的高度是 7 层。当一组 gaussian pyramid 被建立后，抽取倒数第三层的图片作为下一组 gaussian pyramid 的顶层图片，依次得到后面所有组数的 gaussian pyramid。同样，组数也是 SIFT 算法中的一个参数，Lower 论文中组数大小为 8 组。一组高斯金字塔如图 2.1 所示：

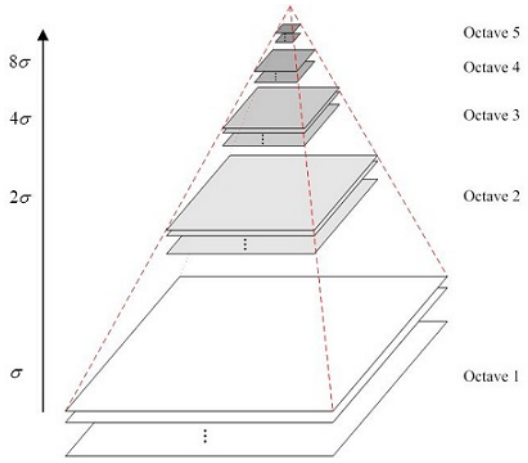

图 2.1　图像的一组高斯金字塔

上面在介绍 gaussian pyramid 时，提到了尺度空间（scale-space），scale space 是 SIFT 算法中一个非常重要的概念。scale space 最早是由 Iijima[47] 于 1962 年提出的，后经 witkin[48] 和 Koenderink[49] 等人的推广逐渐得到关注，在计算机视觉邻域使用广泛。它的基本思想是，使用一个尺度参数，然后得到物体在不同的尺度参数下一段图像序列，最后再获取的序列上进行斑点检测或者是轮廓提取。尺度空间模拟了物体在人的视线中有近到远的消失。尺度理论的数学表达式如公式 2.2 所示，公式中，高斯模板的维度使用 m 和 n 表示，图像的像素位置使用 x 和 y 表示，尺度空间因子使用 δ 表示。

$$L(x,y,\delta) = G(x,y,\delta) * I(x,y) \tag{2.2}$$

其中，* 表示卷积运算，G(x,y,δ) 为高斯函数，

$$G(x,y,\delta)=\frac{1}{2\pi\delta^2}e^{-\frac{(x-m/2)^2+(y-n/2)^2}{2\delta^2}} \tag{2.3}$$

2. 高斯差分金字塔建立

如果将 gaussian pyramid 中上下相邻的两张图片做减操作，即将两张图片的像素值相减，那么就可以得到高斯差分金子塔（DOG），SIFT 算法中，极点的检测实际上是在 DOG 的基础上进行的，目的在于减少检测的运算量。DOG 如图 2.2 所示：

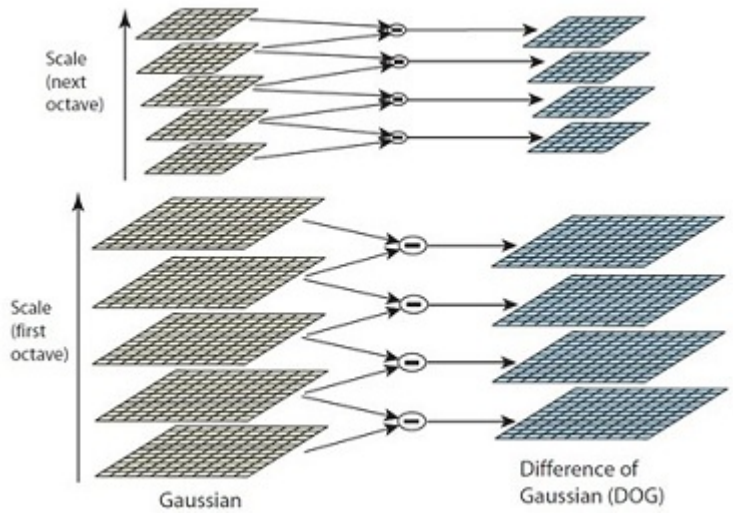

图 2.2　高斯差分金子塔

这里更加仔细的解释一下为什么使用高斯差分金子塔并不会影响提取结果的正确性，在图像中最稳定的特征对应着尺度归一化的高斯拉普拉斯（简称 gauss-laplace）函数 $\delta^2\nabla^2G$ 的极值。而高斯差分函数 (gauss-difference) 与尺度归一化的高斯拉普拉斯函数 $\delta^2\nabla^2G$ 存在等式关系，关系推到如下：

$$\frac{\partial x}{\partial\delta}=\delta\nabla^2G \tag{2.4}$$

利用差分近似代替微分，则有：

$$\delta\nabla^2G=\frac{\partial x}{\partial\delta}\approx\frac{G(x,y,k\delta)-G(x,y,\delta)}{k\delta-\delta} \tag{2.5}$$

因此就有：

$$G(x,y,k\delta)-G(x,y,\delta)\approx(k-1)\delta^2\nabla^2G \tag{2.6}$$

从上面的公式推导，我们就可以看出高斯拉普拉斯函数和高斯差分函数是常数关系，两者极值关系如图 2.3所示，最下方的曲线为 DOG 函数，上方为高斯拉普拉斯算子：

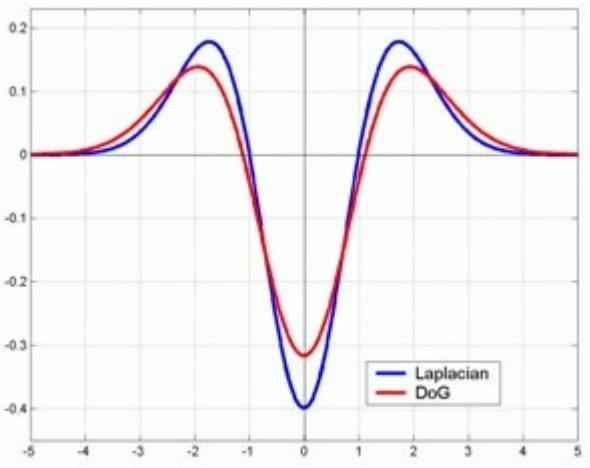

图 2.3　gauss-laplace 和 gauss-difference 极值关系比较

因此，我们就可以使用 gauss-difference 替代 gauss-laplace 进行图像特征点检测，公式如下所示：

$$D(x,y,\delta)=\left(G(x,y,k\delta)-G(x,y,\delta)\right)*I(x,y)=L(x,y,k\delta)-L(x,y,\delta) \qquad (2.7)$$

3. 极点检测

因为 SIFT 中的特征点是从高斯差分金字塔上的极值中得到的（这里先强调一下，不是所有的极值点都是特征点，后续会有专门的步骤将一些噪声大的极点删除点），因此需要在 guass-difference 上对极值的判断，这里的判断是指将需要判断是否为极值的像素点和它为同一层像素点的相邻的 8 个点以及上下两层共 18（9*2）点进行比较，看是否为极值。具体如图 2.4 所示：

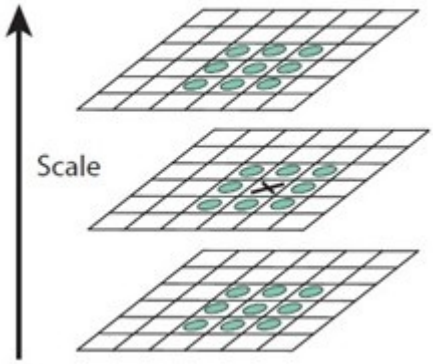

图 2.4　高斯差分图像上的极点检测

4. 关键点精确定位

在上一点中，本文强调了，不是所有在高斯差分金字塔上检测到的极值点都是关键点，因为整个检测过程是在一个离散空间上进行的，因此检测出

来的极值点不是严格的极值点，图 2.5解释了为什么检测出来的极值点不是严格的极点。在这种情况下，就需要对获取到的极值曲线进行子像素插值（Sub-pixel Interpolation），目的在于获取更加严格的极值点，以保证特征点的质量。在插值操作完成后，为了使所有关键点表现的更加平滑，需要对关键点进行拟合，因为如果两个相邻的关键点的差值很大，会造成比较大的噪声，这样的特征点效果不好。拟合具体表现就是对高斯差分函数进行 Taylor 展开式，拟合后的函数如 2.8所示。根据拟合后的 guassin-different 函数，就可以设置阈值进行过滤，去掉一些不满足要求的特征点，使得提取的特征点更加稳定。

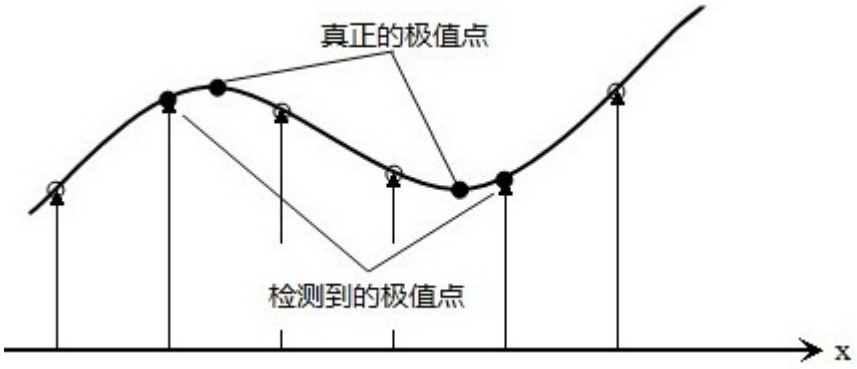

图 2.5　检测出来的极点和严格意义上的极点比较

$$D(X) = D + \frac{\partial D^T}{\partial X}X + \frac{1}{2}X^T\frac{\partial^2 D}{\partial X^2}X \tag{2.8}$$

5. 关键点方向分配和描述

当图像的所有关键点被检测出来后，接下来就要为关键点分配一个主方向，主方向的确定首先要统计关键点以邻域内的所有关键点的梯度的模值和方向，计算公式如2.9所示，其中关键点的尺度空间值表示为 L，然后将以上得到的梯度按方向的进行归类（这里的类别共有 36 种，从 0 到 360 度，每 10 度为 1 类，比如，10 度，20 度，30 度，...，360 度），统计每一类中总模值的大小，最大的模值的方向就为主方向，同时也会选取总模值大于主方向的模值的 80% 为辅方向，辅方向的建立是为了增强关键点的鲁棒性。上述过程最终会形成一个直方图，具体如图2.6所示。

$$m(x,y) = \sqrt{L(x+1,y) - L(x-1,y)^2 + L(x,y+1) - L(x,y-1)^2} \tag{2.9}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y - L(x-1,y)}\right) \tag{2.10}$$

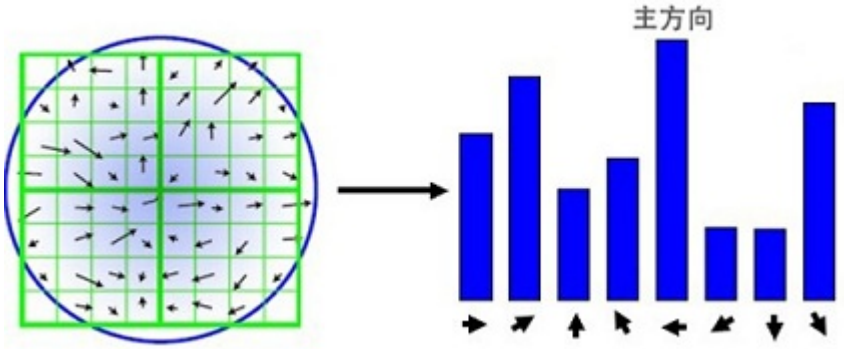

图 2.6　关键点方向直方图

在关键点的主方向被确认后，SIFT 算法还没有结束，因为计算机处理的只有数字信息，因此必须采用严格数字符号将关键的信息描述出来，SIFT 算法中采用长度为 128 的一维向量对一个关键点进行描述，向量的含义为以关键点为中心，4*4 窗口区域内（每个窗口为一个关键点），计算所有关键点在 8 个方向上的梯度，共计 128 个梯度信息。

在上述 5 个步骤中，构建高斯金字塔和极点检测是最为耗时的两个。构建高斯金字塔时需要对每张图片进行高斯滤波，高斯滤波是一个时间复杂度为 $O(N^3)$ 的操作，这里的 N 与图像大小成正比。图 2.1中仅显示了一组金字塔，而 SIFT 算法往往需要构建多组金字塔，使得构建高斯金字塔的时间复杂度达到 $O(O*I*N^3)$，其中 O 为高斯塔的组数，I 为高斯塔的层数。极点检测则是在整个金字塔组上进行的，相当于在一个维度为 4 的空间（组，层，长，宽）中搜索极值点，它的时间复杂度为 $O(O*I*N^2)$。SIFT 算法流程中既包含高斯滤波和图像缩放等通用性较强的操作，也含有构建高斯金字塔、极点检测、消除边缘响应等通用性较差的操作。它的空间复杂度和时间复杂度都比较高，当有海量的图片需要进行特征提取时，占用内存较多，所需的处理时间也会很长。这也是我们使用 Spark 去加速 SIFT 算法的原因。

只有充分理解 SIFT 算法原理之后，才能比较好的在 Spark 上实现大规模特征提取工作。

2.3　Spark 核心技术原理

在这一小节中，将会对 Spark 中三个核心技术原理进行分析，它们分别是内存编程原理，任务调度原理，性能优化。这三个技术在本文的设计起非常重要的地位，只有将这些技术理解透彻后，才能自如的解决开发过程中遇到的问题。

2.3.1　Spark 内存编程原理

Spark 区别于其他的大数据处理框架，比如 Hadoop[50], 在于它是一个内存计算的大数据处理框架，它可以将处理的中间结果保存在内存中，后续的计算可以在此基础上直接运行，提升了执行的效率。那么，这么多的数据保存在内存中，这就必须要有基于分布式内存的数据结构抽象，让开发人员基于该数据结构进行

操作，只要操作这些数据，它们就是位于集群的内存中的。上面说的内存抽象数据结构就是 RDD(Resilient Distributed Dataset)，RDD 是 Spark 最核心的概念，接下来将从 RDD 的底层实现原理，RDD 的操作类型，RDD 的缓存原理，RDD 的依赖关系和 DAG 的生成这几方面分析 RDD。

2.3.1.1 RDD 的底层实现原理

RDD 中包含多个分区，分区是构成 RDD 的最小单位，同时分区也是和任务并行联系在一起的，一个分区就对应着一个并行执行的任务。整个过程如图 2.7所示：

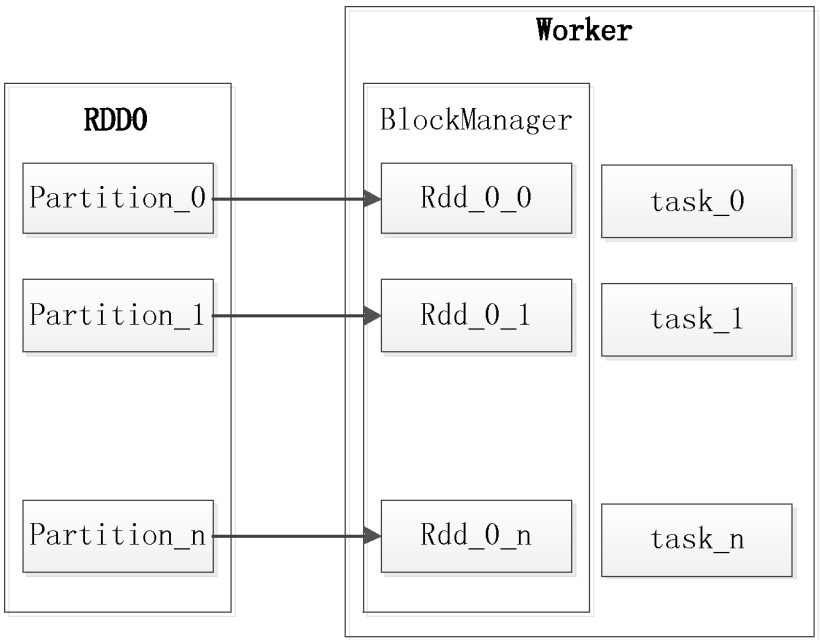

图 2.7 RDD Partition 的存储和计算模型

2.3.1.2 RDD 的操作类型

RDD 提供的操作可以分为两种类型，分别是 transformtion(转换) 和 action(动作)，表2.1和表2.2分别列出了常用的转换操作和动作操作。所有 transformation 类型的算子被调用时，Spark 不会马上进行算子的执行，Spark 会使用一个有向图（DAG）将这些操作记录下来，当一个 action 算子被调用时，那么在这之前的所有 transformation 操作全都被执行。这种设计会让 Spark 运行得更加高效率，因为可能在动作操作时，被操作的数据集会变得比较小。

2.3.1.3 RDD 的缓存原理

RDD 的特点在于它是一个内存数据结构，所有对它的操作是在内存中完成的，这也是为什么 Spark 在迭代次数多的任务中计算的速率要远快于 Hadoop。在 Spark 中，某个数据集（RDD）会被后续操作使得到，或者是整体使用的频率十分高，这时候就可以将该数据集据保存在内存中，这个操作在 Spark 中称为数据的内存持久化。在代码实现时，持久化一个 RDD 是通过调用 persist() 或者是 cache() 函数实现的。persist() 和 cache() 的在 Spark 源码中的定义如下：

表 2.1　RDD 转换操作

转换	含义
map(func)	对所有分区上的数据都执行 func 操作
filter(func)	具有数据过滤的功能，对所有分区上的数据使用 func 条件进行过滤
flatMap(func)	这个函数和 map 是具有同样功能，区别在于它处理完后会将结果变成一个一维数组
mapPartitions(func)	该函数和 map 函数功能相似，只是这个函数的作用目标是整个分区的数据
mapPartitionWithSplit(func)	该函数和 mapPartitions 函数功能相似，但是作用的数据是带有分区号的
sample(withReplacement,fraction,seed)	对目标的数据集进行采用，fraction 是采用比例，seed 为随机种子
distinct(numTasks)	对分区上的数据进行去重操作

表 2.2　RDD 动作操作

动作	含义
reduce(func)	通过函数 func 对数据集中的所有元素进行特定的归并操作
collect()	将所有的分区的运算结果收集到 Driver 上
count()	返回数据集的个数
first()	返回数据集的第一个元素
take(n)	这个算法和 collect 有所区别，collect 是返回所有结果，take 是返回前 n 个结果
takeSample(withReplacement,num,seed)	在所有分区的数据上进行采用
saveAsTextFile(path)	将运算结果以文本的方式保存到 HDFS 中
saveAsSequenceFile(path)	将运算结果以序列化的方式保存到 HDFS 中
countByKey() 根据 key 信息统计记录的条数	

```
/*通过*函数进行内存缓存persistrdd*/
def persist(): this.type = persist(StorageLevel.MEMORY_ONLY)
/*通过*函数进行内存缓存cacherdd*/
def cache(): this.type = persist()
```

假设首先进行了 RDD0→RDD1→RDD2 的计算作业，如果 RDD1 已经被缓存在内存中，那么后续在进行 RDD0→RDD1→RDD3 的计算作业时，就不需要从头开始，直接从 RDD1 往下执行作业即可，因此计算速度得到了很大的提升，具体过程如下图2.8所示：

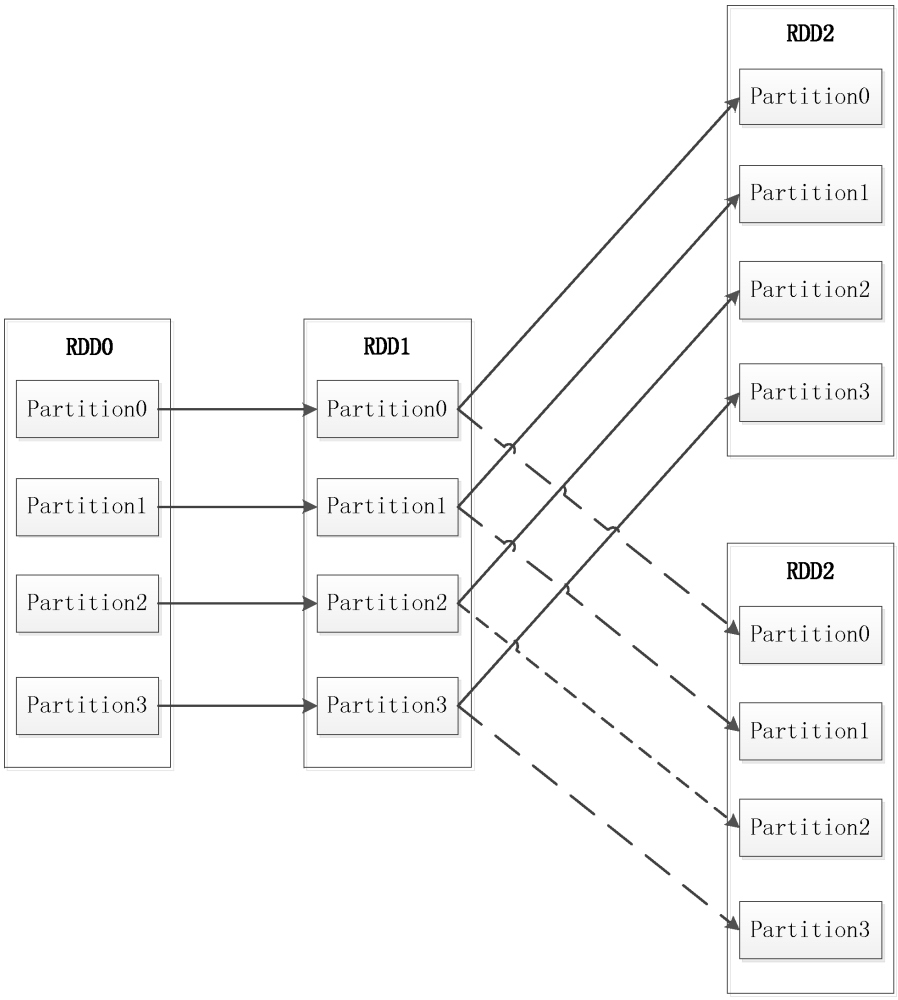

图 2.8　RDD 缓存加速原理

因为 RDD 是直接保存在内存中，因此 RDD 很可能会因为某些原因而丢失，但是 Spark 中有完整的 RDD 容错机制来保证 RDD 在内容丢失的情况下重新计算。

2.3.1.4　RDD 的依赖关系和 DAG 的生成

一个 Spark 应用中，不同的 RDD 间存在依赖关系，依赖关系可以归结为两种，它们分别是为窄依赖 (narrow dependency) 和宽依赖 (wide dependency)，窄依赖和宽依赖的定义如下：

- 窄依赖，窄依赖是指父 RDD 的分区结果和子分区使用是一个 1 对 1 的关系，如图2.9左边部分显示
- 宽依赖，宽依赖是指父 RDD 的分区结果和子分区使用是一个多对 1 的关系，如图2.9右边部分显示

在 Spark 应用开发中，同一个 RDD 一般都会被多次转换，而新的 RDD 会伴随着转换操作而产生，这些转换操作最后会形成一个有向图（Directed Acyclic Graph，简称 DAG)。这个 DAG 记录着 RDD 间的依赖关系，这些依赖关系用于判断一个

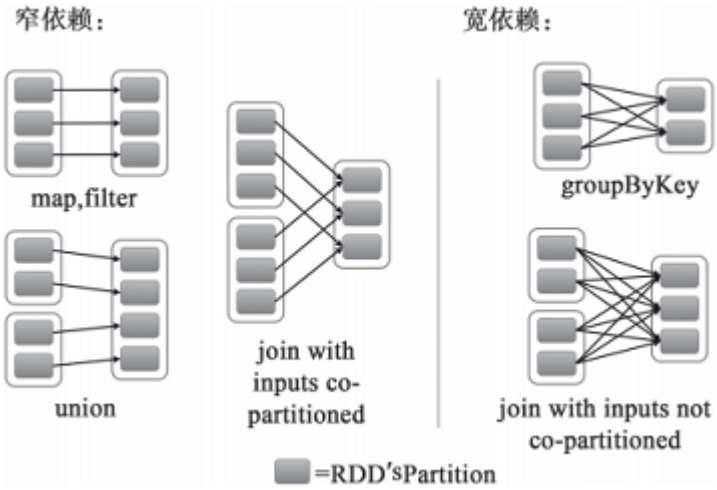

图 2.9　RDD 窄依赖和宽依赖对比

RDD 是否可以开始被执行，因为如果一个 RDD 和它的父亲 RDD 间是宽依赖关系，那么在父亲 RDD 没有执行完成之前，该 RDD 是不能开启执行的。Spark 会根据 DAG 对算子进行 stage 的划分，划分的依据是，rdd 间是窄依赖关系的划分到一个 stage 中，rdd 间是宽依赖的划分到不同的 stage 中。一个 stage 中的 task 可以并行执行，不同 stage 的 task 则要顺行执行。

2.3.2　Spark 任务调度原理

Spark 任务调度框架主要由三个模块支撑起，它们分别是有向图调度器（DAGScheduler），后端调度器（SchedulerBackend）及任务调度器 TaskScheduler。首先 DAGScheduler 分析用户提交的应用，根据依赖关系建立 DAG，然后将 DAG 划分为不同的 Stage，而 stage 中包含所要执行的 tasks。SchedulerBackend 负责整个集群交互，收集可用的资源信息，然后将这些信息上报 TaskScheduler。TaskScheduler 将从 SchedulerBackend 获取到的资源信息以及从 DAGScheduler 生成的 tasks 进行一个资源的分配，将 tasks 分配的合适的资源上面。整个流程如图2.10所示：

2.3.2.1　DAGScheduler

DAGScheduler 的功能是对应用程序提交的 task 根据依赖关系进行 stage 的划分，将窄依赖关系的 task 划分到一个 stage 中，然后将 task 一个集合的形式进行打包，形成 taskset，再将 taskset 传递给 TaskScheduler。

2.3.2.2　SchedulerBackend

SchedulerBackend 负责和整个集群打交道，收集集群的资源信息，并且将这些信息上报给 TaskScheduler。SchedulerBackend 中有两种调度方式，分别是粗粒度和细粒度两种，运行时具体是哪一种取决于集群的运行模式。粗粒度下，tasks 是按照 Executor 进行分配的，而在细粒度下，task 是按照线程进行分配的。

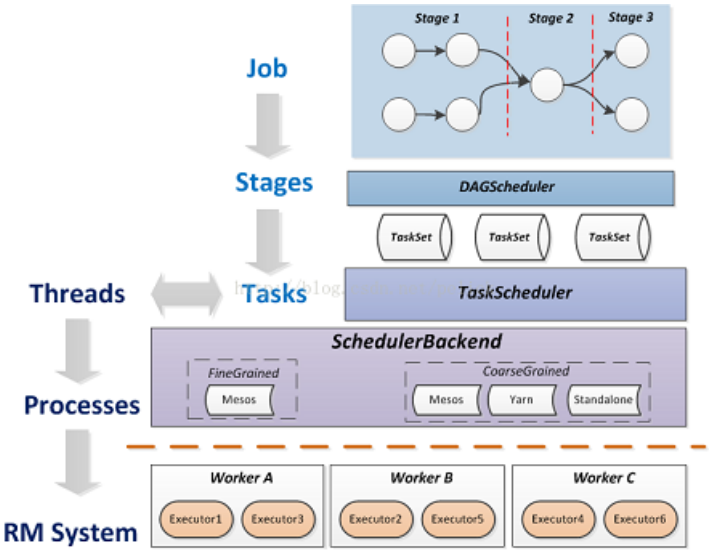

图 2.10　Spark 任务调度框架

2.3.2.3　TaskScheduler

TaskScheduler 的任务是负责将从 DAGScheduler 传递上来的 Tasks 分配到合适的工作节点上面，对于 Taks 的分配策略有两种，分别是 FIFO(先进先出) 以及 FIFA(公平调度)。TaskScheduler 中存在两个列表，一个是可以被 launch 的 tasks 列表，另外的是可以被 use 的 worker 列表，TaskScheduler 如同一个管家，按照列表的顺序，吩咐相应 worker 去处理相应的 tasks。但是如果一直按照这种方式，可能是某个 worker 或者是某几个 worker 会经常处于忙碌状态（因为他们都比其他 work 的计算能力强，一直在” 抢” 任务），而大部分 worker 一直处理闲状态。所以为了避免这种情况，TaskScheduler 会对 Worker 进行 shuffle 操作，任意挑选 worker 放在 worker 列表的前面。

2.3.3　Spark 性能优化原理

在本小节中，将介绍 Spark 的性能优化原理，因为 Spark 是处理大数据的框架，因此性能优化是非常有必要的。接下来将从几个方面分析 spark 的性能问题

2.3.3.1　调度和分区优化

在 Spark 中，一个分区是对应着一个 task 的，即一个分区是一个并行的单元，那么是不是分区的数量越多，一个 Spark 应用就会运行的越快呢？其实并不是这样，不是分区数越多，运行的速度的越快，因为当分区数量十分大时，task 的调用开销也会变得很大，有可能出现 tasks 的调度时间比任务运行的总时间还要长。所以，在制定分区数量时要综合考虑集群系统的资源和被处理作业的数据量。

在分区优化问题中，还有一个倾斜 (skew) 的问题。倾斜可以分为数据倾斜和任务倾斜，两者存在关系，数据倾斜会引发任务倾斜。

1. 数据倾斜

数据倾斜是指数据被划分的不均匀，一个 task 被分配了很多的处理数据，有

可能比所有 task 分配的还要多，在这种情况下就会导致负载的不均衡，下面是一些常见的引发 Spark 在出现数据倾斜的原因:

- key 数据分布不均匀，因为 spark 中默认是按照 Hash 方式进行分区，如果某个 key 的数据特别多，就会造成分到某个 key 上的数据特别多，最后造成某个分区的数据特别多，也就出现了数据倾斜。
- 结构化数据表设计问题
- 某些 SQL 语句产生数据倾斜

2. 任务倾斜
产生任务倾斜的原因较为隐蔽，有可能是服务器架构的原因，有可能是 JVM 的原因，有可能是线程池的原因，也有可能是业务本身的原因。比如在本文设计中就发现，即使是两个任务的处理数据的总大小一样，但是仍然会产生任务倾斜，这和处理问题的具体场景有关系，在 SIFT 算法中，算法的时间复杂度和图片的尺寸大小有关系。

2.3.3.2 内存存储优化

因为 spark 是内存计算的大数据处理框架，因此内存存储优化也是十分重要的内容。内存调优过程中，有三个方向值得考虑，分别是 JVM 调优,OOM(OutOfMemoryError) 问题调优及磁盘临时目录空间优化。

1. JVM 调优
不同的 JAVA 对象都有一个对象头，这些信息有时候比数据本身的信息还要大，比如只有一个 Int 属性的对象。还有一些链式结构，它们会包含一些指针信息。因此在开发的过程要选好数据类型和数据结构，尽量减少一些链式结构的使用；减少对象的嵌套；在 key 的类型上，可以考虑使用数字来替代字符串。
2. OMM(OutOfMemoryError) 问题调优
在 spark 开发过程中，内存溢出 (OutOfMemoryError) 是一个会经常会遇到的问题，发生内存溢出的原因是应用程序在索求的内存空间已经超过了虚拟机的分配的堆空间。解决这类问题有两种思路，一种是减少 App 的内存占用空间消耗，另一种是增大内存资源的供应。具体设计时，可以查看程序中是否有循环创建对象的地方，程序中应该减少这种代码的存在，或者可以调整 JVM 中的 Xmx(最大堆) 和 Xms(最小堆) 参数的大小。另外，还要查看在做 Shuffle 类操作符时是否创建的 Hash 表过大，在这种情况下，可以尝试增大应用程序的分区数量以减少每个任务处理的数据量。

2.3.3.3 磁盘临时目录空间优化

Spark 在进行 Shuffle 的过程中，中间结果会写到 Spark 在磁盘的临时目录中，如果临时目录过小的话，会造成存储空间不够的异常，在这种情况下可以配置多个盘块来扩展 Spark 的磁盘临时目录，让更多的数据可以写到磁盘，加快 I/O 速度。

2.3.3.4 网络传输优化

Spark 是大数据处理框架，因此集群中数据传输也是影响应用性能的重要因素。如果可以减少数据传输的次数或者是传输的大小，可能会极大的提高应用的性能。下面将分析 spark 中数据传输优化的场景。

1. 大局部变量的传输

   在 spark 的分布式处理算子中，如果使用了不是算子内部的变量，那么 Spark 会为每个 task 生成一个这样的变量，那么如果这个变量十分大，比如是一个大数组，而且执行的任务又特别多，那么网络的传输耗费就会特别大。下面就是一个例子的展示, 其中 factor 是一个变量，它在 map 任务中被使用:

   ```
   val factor = 3
   rdd.map(num => num*factor)
   ```

   面对这种情况，我们可以采用 Spark 中 Broadcast 或者是共享变量这两种方式来解决大外部变量传输的问题，通过这两种方式，Spark 只会为 Worker 生成需要的外部变量，而不是为每个 task 生成一个外部变量，减少了局部变量的传输开销。

2. 大结果收集

   在 spark 的开发中，会经常用到 collect 操作，collect 操作将各个分区的结果收集到一个数组，返回到 driver。如图2.11如果收集的结果过大，就会拖慢应用的执行时间，甚至造成内存溢出。面对这种情况，如果收集的结果过

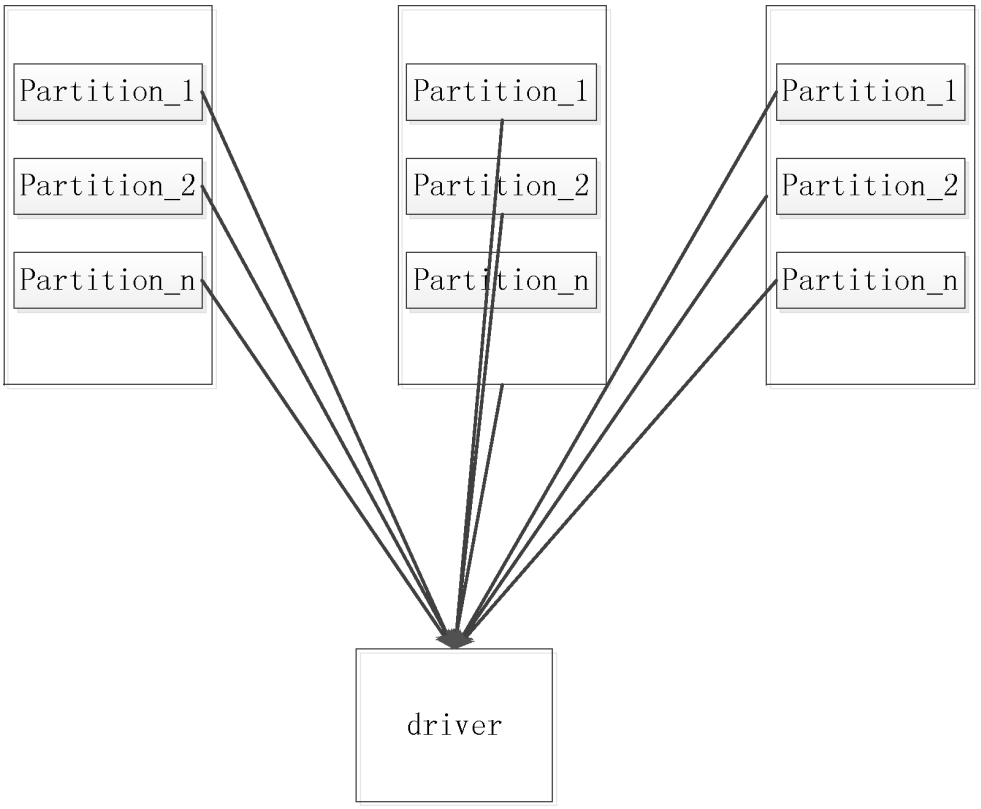

图 2.11　collect 操作收集所有分区

大，可以将数据分布式的存储在 HDFS 或者其他的分布式持久化层，这样可以减少单机数据的 I/O 开销和单机内存的存储压力。

2.4 分布式存储原理

在大数据处理场景下，需要的处理的数据集往往是很大的，比如 100T，那么使用单节点的存储方案是基本不可能的，无论是数据的存储空间或者是数据的安全方面，单节点是无法适用需求的。因此，需要一种集群式的的存储方案来应对大数据的场景下的数据保存，HDFS 分布式文件是根据 google 的三大论文中的 The google fileSystem 文章开发出来的。HDFS 是一个 Master-Slave 的架构，里面有 Namenode，DataNode 及 SecondaryNameNode 三种角色，NameNode 负责管理数据的元数据信息，DataNode 负责数据的 data 数据的保存，SecondaryNameNode 负责数据的备份。HDFS 是按照 Block 为单位进行数据存储的，在 HDFS 中一个 Block 的大小通常为 64M，从这里就可以看出 HDFS 中的 Block 的大小比我们 Linxu 的文件系统的 Block 要大很多，这也是为了在大数据场景下提升数据的读写效率。在写入数据的时候，如果被写入文件如果大于 64M，HDFS 会进行数据的划分，按照块的大小进行划分，那么一个文件就可能被划分为几个块，进行分布式存储。同样在读取数据的时，hdfs 客户端将文件名以及字节的偏移发送给 NameNode，NameNode 根据字节偏移量转化成 Block 偏移，然后将给偏移量返回客户端，客户端根据该偏移量信息到 DataNode 上进行数据的读取，后续的操作也不需要经过 NameNode。

2.5 GPU 特征提取原理

因为在实验数据比较上，我们和 GPU 下的特征提取进行了比较，因此，在这小节中的介绍 GPU 的相关原理。GPU 最开始的时候是应用在图像显示领域的，作为显卡使用，后面因为其出色的并行加速能力，逐渐应用于科学计算，机器学习及深度学习的领域中。我们通过是将 CPU 上相关需要加速的算法改写成 GPU 版本，是程序中可以并行的代码区域，在 GPU 上以多线程方式并行执行，大大提升了程序的运行时间。GPU 之所以能够比 CPU 拥有如此大的并行加速能力，原因在 GPU 上的计算单元要比 CPU 多出很多，GPU 上的计算单元可以划分为三个维度，分别设计 Grids，Blocks 以及 Threads，大量的 Threads 组成一个 Block，然后大量的 Blocks 组成一个 Grids，最后由大量的 Grids 组成强大的计算能力。

2.6 本章小结

本章分析了本文工作的一些技术原理，首先分析 SIFT 算法的基本数学原理，主要针对 SIFT 算法的 5 个主要步骤进行分析，分析 5 个步骤的数学原理及在 SIFT 算法中的作用，然后讨论了 SIFT 算法的中最为耗时的步骤。在介绍完 SIFT 算法原理之后，本章节对 Spark-SIFT 的大数据模块进行原理分析，首先分析了 park-SIFT 的计算引擎 Spark 主要分析 Spark 核心技术原理，包括 Spark 的内存编程原理，任务调度原理，性能优化原理。接着又分析了充当 Spark-SIFT 系统存

储层的 HDFS 文件系统的一些核心原理。最后，介绍了作为本文设计工作对比的 GPU 特征提取的一些基本原理。

第三章　大规模特征提取系统 Spark-SIFT 实现

在本章节中将详细介绍整个大规模特征提取系统 Spark-SIFT 的设计和实现。首先会介绍系统的整体框架和数据流程，接着会针对系统中的各个模块进行分析，主要包括功能模块，存储模块，容错模块，最后介绍系统的部署以及正确性验证。

3.1　系统架构与数据流程

大规模图像库特征提取系统 Spark-SIFT 框架图如图3.1所示：图像库进行特征

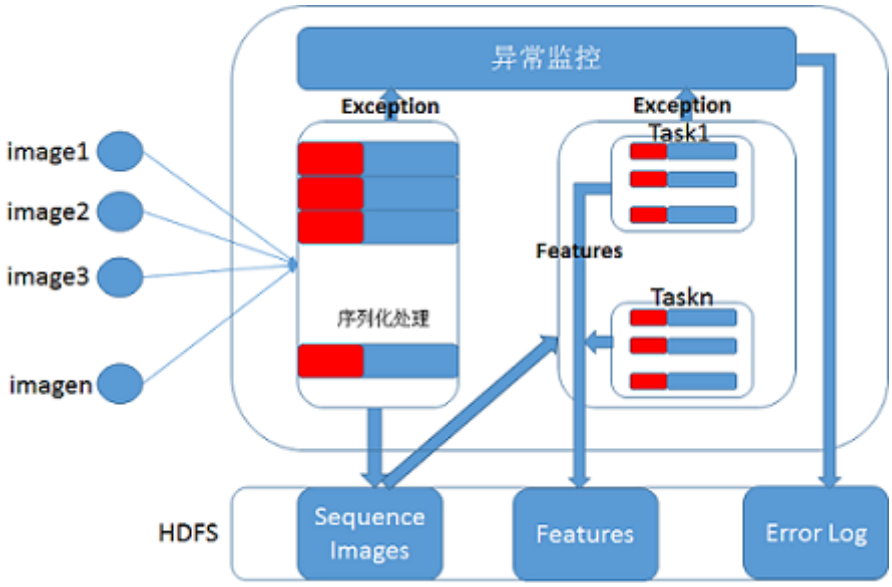

图 3.1　大规模图像库特征提取系统 Spark-SIFT 系统框架

提取前，先经过序列化处理模块进行序列化处理，经过序列化模块之后，一张图片转化单条记录的形式，最后将所有的记录序列化保存到分布式文件系统 HDFS 中。当进行特征提取时，特征提取功能模块读取 HDFS 中序列化图片记录，将其分发到各个任务上，任务运行在整个集群上，每个任务上通过 Spark-SIFT 算法进行特征提取，完成特征提取后，每个 task 直接将提取出来的特征保存到 HDFS 中。Spark-SIFT 系统中的异常监控模块收集特征提取作业运行过程中出现的异常信息，将异常信息写到特定的异常日志中。

从软件架构的角度看整个系统，系统的框架图如图3.2所示，本文的主要工作在于 Spark 的库层和应用层。本文实现了 Spark 下的图像处理基础库 Spark-imageLib，这个库包含了 spark 的图像的表示，读取操作及一些基础的图像处理接口，比如图像的灰度化，图像的缩放，图像的模糊等；这个库和 spark 的其他库，比如分布式图计算处理库 GraphX，机器学习库 MLib 的地位是等价的，我们已经将该库嵌入到 spark 的生态系统中。此在这个库的基础上，开发了 Spark-SIFT 算

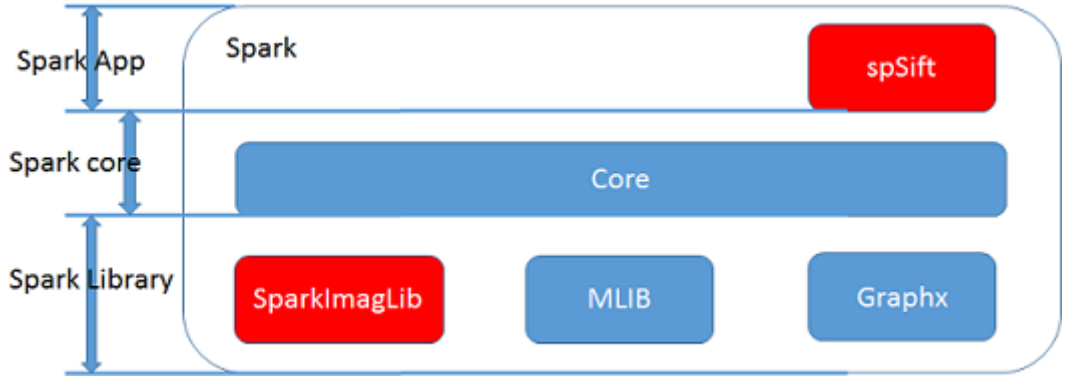

图 3.2　Spark-SIFT 软件架构

法，该算法在 Spark 应用中被调用，实现分布式的特征提取工作，图中的 spSift 就是特征提取应用，它位于 Spark 的应用层。

3.2　系统模块设计

在本小节中，会详细介绍每个模块的设计及实现，它们分别是 spark 图像基础库 Spark-imageLib，Spark-sift 算法，序列化模块，存储模块以及系统容错模块。

3.2.1　spark 图像基础库 spark-imageLib

Spark-imageLib 图像处理库主要包括图像的表示，图像的读取，特征点的表示。

3.2.1.1　图像表示

在图像表示中，本文设计主要用到了三个类，分别是 SpImage，SpSingle-BandImage 及 SpFImage，它们间的继承关系如图3.3所示：SpImage 为最基础的抽象类，不会直接被使用。SpSingleBandImage 为 SpImage 的子类，它也为抽象类，SpSingleBandImage 表示单通道类型的图像。SpFImage 类为 SpSingleBandImage 的子类，也是本文中实际用来在 Spark 下表示图像的类。这三个类都有一个共同的属性变量，就是 serialVersionUID，因为本文是一个分布式的图像处理框架，必然会涉及到处理图像文件的网络传输，因此表示图像的类必须是序列化的，这样才能使得图像可以在 Spark 这个分布式框架下处理。这里的序列化采用的 java 的序列化方式，因为序列化必然会存在一个反序列化的过程，比如从 Master 将图像序列化传送的 Worker 上，然后 Worker 将接受到的内容反序列化得到原始图像，为了避免两端实体类的内容不一样，就使用了 serialVersionUID 作为衡量的标准。当 JVM 获取传来的字节流时，先读取 serialVersionUID，然后和本地的进行比较，根据比较的结果判断是否可以进行序列化。

考虑传输过程中的耗费代价，因此设计 SpFImage 数据结构时要十分精简，在成员变量中，只设置了一个二维数组 pixels，这个数组就是保存整张图片的像素信息。因为考虑到高斯模糊时候的精度问题，所以 pixels 采用了 float 类型。SpFImage 的构造函数是比较灵活的，它允许传递一维数组或者是二维数组，数组的类型是 byte 或者 float，对于不同的参数类型，本文都会自动进行转换。

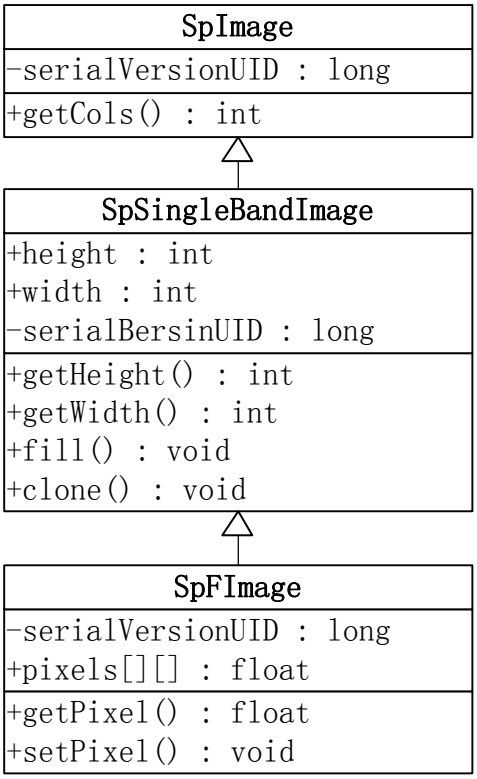

图 3.3　Spark 下图像表示

表3.1列出三个类的主要功能函数:

表 3.1　SpFImage 主要功能函数

函数名	所属类	功能
abs	SpFImage	对图片的像素点进行绝对值操作
addInplace	SpFImage	将图片的像素进行加操作
subtractInplace	SpFImage	将图片的像素进行减操作
multiplyInplace	SpFImage	将图片的像素进行乘操作
divideInplace	SpFImage	将图片的像素进行乘操作
fill	SpFImage	对图片进行填充操作
getDoublePixelVector	SpFImage	将图片像素以一维向量的形式返回
getPixel	SpFImage	获取图片某个位置上的像素值
setPixel	SpFImage	对图片某个位置的像素进行赋值
threshold	SpFImage	对图片像素进行阈值判断

3.2.1.2　图像的读取

在图片读取这个模块中，主要使用到 SpImageUtilities 类型，该类提供了从字节流到 SpFImage 的转换接口，该类的主要成员函数表3.2所示。

表 3.2　SpImageUtilities 主要功能函数

函数名	所属类	功能
readF	SpImageUtilities	将字节流转化成 SpFImage
createFImge	SpImageUtilities	将 BufferedImage 类型的图片转换成 SpFImage 类型
createBufferImage	SpImageUtilities	将 SpFImage 类型图片转换成 Buffer-Image 图片
checkSize	SpImageUtilities	对图片的大小进行检查

因为图片库是保存在 HDFS 中的，因此需要用特定的 HDFS 接口进行读取，在读取完之后在使用 SpImageUtilities 接口进行转换。在 HDFS 读取接口中，可以通过三个接口进行图片的读取，它们分别是，sequenceFile，binaryFiles 及 objectFile 函数。sequenceFile 是 spark 下用来读取序列化文件的接口，具体定义如下：

```
def sequenceFile[K,V](path:String,KeyClass:Class[K],valueClass:
    Class[V],minPartitions:Int):RDD[(K,V)]=withScope{}
```

sequenceFile 保存的是 Writable 对象，换句话讲，key 和 value 也必须是 Writable 类型的, 表3.3列出了常见类型的 Writable 对应类型。

表 3.3　Writable 对应类型

Scala 类型	Java 类型	Writable 类型
Int	Integer	IntWritable
Long	Long	LongWritable
Float	Float	FloatWritable
Double	Double	DoubleWritable
Array[Byte]	byte[]	BytesWritable
String	String	Text
List[T]	List<T>	ArrayWritable
Array[T]	T[]	ArrayWritable

本文设计中，采用 sequenceFile 读取图片的主要代码如下：

```
val fn_rdd = sc.sequnceFile(imageSEQ_path,classOf[Text],classOf
    [BytesWritable],task_size.toInt)
.map({case (fname,fcon)} => {
  var datainput:InputStream = new ByteArrayInputStream(fcontext
      .getBytes)
  var img = SpImageUtilities.readF(datainput)// 读取图片的像素矩阵
...
})
```

采用 binaryFiles 接口进行读取时，返回的是一个 key-value 类型的 RDD，key 为文件名，而 value 则是文件的内容的字节流，binaryFiles 的定义如下：

```
def binaryFiles(path:String,minPartitions:Int =
   defaultMinPartitions):RDD[(String,PortableDataStream)]=
   withScope{}
```

使用 binaryFiles 接口读取图像的主要代码如下：

```
sc.binaryFiles(initImgs_path_hdfs,task_size.toInt).map(f => {
    val fname = new Text(f._1.substring(prefix_path_hdfs.
       length,f._1.length))// 获
       取features key
    val bytes = f._2.toArray()

    var datainput:InputStream = new ByteArrayInputStream(bytes
       )
    var img = SpImageUtilities.readF(datainput)
    ...
}
)
```

objectFile 读取的是对象文件，保存的时候以什么对象类型保存，就以什么类型读取读出，该函数看似是对 SequenceFile 的简单封装，它运行存储只包含值的 RDD，两者的区别在于 objectFile 保存对象时是采用 java 序列化的方式，SequenceFile 采用的则是 hadoop 的序列化方式。objectFile 函数定义如下：

```
def objectFile(path:String,minPartitions:Int =
   defaultMinPartitions):RDD[T]=withScope{}
```

使用 objectFile 接口读取图像的主要代码如下：

```
sc.objectFile(initImgs_500k_path_hdfs).map(x => {
   val test:SpFImage = x
   ...
})
```

虽然 objectFile 读取图像十分方便，但是如果直接将对象保存到 HDFS 中，储存空间相当大，存储的时间也十分长。

3.2.1.3 特征点表示

在本文设计中，特征点的表示主要涉及到三个模块，分别是 SpKeypoint，SpMemoryLocalFeatureList，SpLocalFeatureList。SpKeypoint 表示一个 sift 特征点，SpMemoryLocalFeatureList 保存整张图片的 sift 特征点，SpLocalFeatureList 为接口，SpMemoryLocalFeatureList 是 SpLocalFeatureList 的具体实现。SpKeypoint 成员结构如图3.4所示：

其中 x,y 代表特征点在图片上的坐标，scale 表示特征点的尺度大小，ori 表示特征点的方向，ivec 数组为特征点的一维描述向量，长度为 128。

SpMemoryLocalFeatureList 数据结构保存 SIFT 算法检测一张图片得到的所有特征点集，因此 SpMemoryLocalFeatureList 具有集合特性，本文设计时，让该类继承 ArrayList，图3.5 为 SpMemoryLocalFeatureList 的 UML 视图。

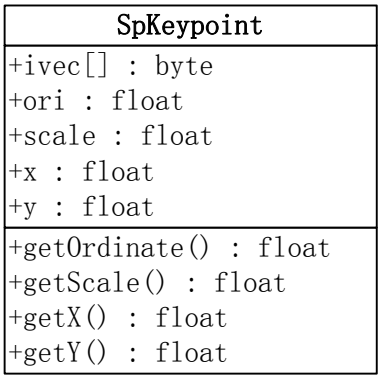

图 3.4　SpKeypoint 数据结构

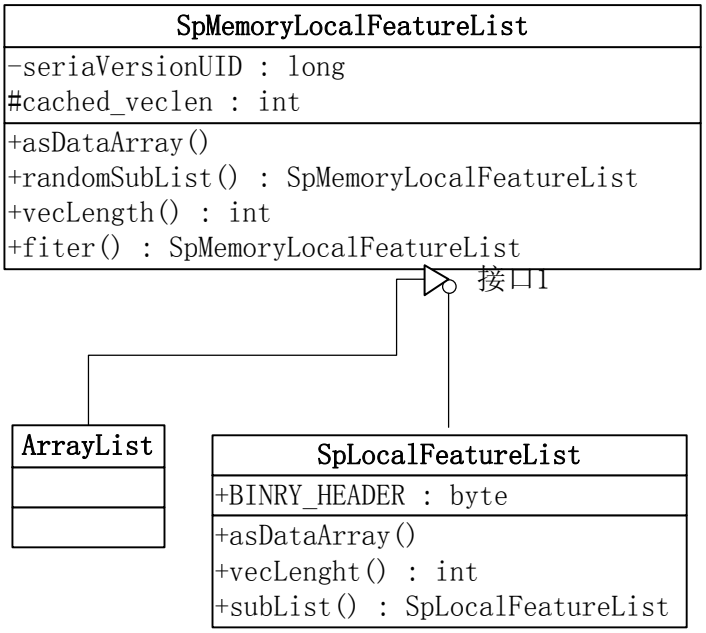

图 3.5　SpMemoryLocalFeatureList 数据结构

3.2.2　Spark-sift 算法

在这小节中将介绍 spark 下怎样使用 sift 算法进行大规模图像库特征提取工作。主要的思想是利用上一小节介绍的 spark-imagelib 来实现整个 SIFT 算法，SIFT 算法在第二章原理分析中已经详细的分析过了，在此不再详述，然后利用 spark 的 Map-Reduce 框架，在 map 函数中调用已经实现好的 SIFT 算法，在提取结束后，将特征向量保存到 HDFS 中，完成整个特征提取的工作。Spark-sift 算法如图3.6所示：

spark-sift 算法的关键伪代码如下：

```
val conf = new SparkConf()//创建spark configure
conf.set()//设置configure
val sc = new SparkContext(conf)//创建spark 运行上下文

rm_hdfs(hdfs_name,kpslibdir+dataset)//删除原有的特征库
```

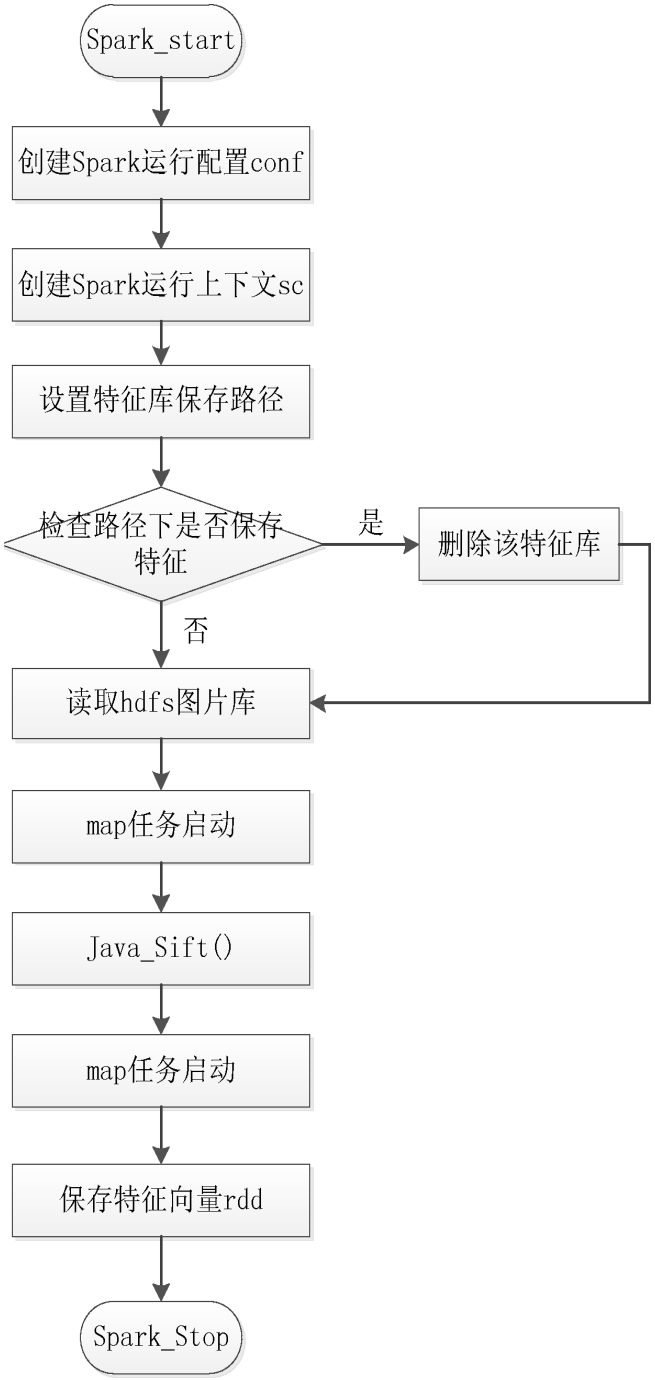

图 3.6　spark-sift 算法流程

```
6  |
7  |//并行执行图片特征提取任务
8  |val fn_rdd = sc.sequenceFile(,
   |    imageSEQ_pathtask_size.toInt).map({
9  |var datainput = new ByteArrayInputStream(fcontext.getBytes)
10 |
11 |var img = SpImageUtilities.readF(datainput)
12 |if(img != null){
13 |    var engine = new SpDoGSIFTEngine()//进行sift 特征提取
```

```
    var kps = engine.findFeatures(img)

    val baos = new ByteArrayOutputStream()
    IOUitls.writeBinary(baos,kps)
    (new Text(fname.toString), new BytesWritable(baos.
       toByteArray))//以key-形式返回提取的特征点，为图片名，为特征点
       集valuekeyvalue
}
})
```

通过上面这段代码，每张图片将被划分到不同的分区中，在每个分区中，执行特征提取任务，然后将得到的特征点以 Key-Value 的形式分布式保存到 HDFS 中，其中 key 为图片的文件名，value 为特征点集的字节流。

3.2.3 序列化模块

因为互联网上图片的体积一般都是 1M 以下，HDFS 以 Block 形式进行存储，Block 的大小是 128M，因此这些文件对于 HDFS 来说都是小文件，众多的小文件存在时，就会造成频繁的磁盘 IO，从而导致读写的性能降低。其实，这种场景在我们生活中也有体会，从 U 盘拷贝一个 2G 的文件和拷贝一个由很多小文件组成的 2G 文件夹，速度是相差很大的。

基于上述情况，本文在整个系统中加入了图片序列化模块，通过该模块将图片转化为 Key-Value 记录的形式，然后将众多的记录合并并且序列化保存到 HDFS 中，后续的处理都是在该序列化后的记录上进行，从而提升 HDFS 读写效率。本文会在第四章的性能优化中对序列化优化方案进行详细分析，在这里，主要介绍代码框架设计。序列化模块流程框架如图所示3.7。

序列化模块关键伪代码如下:

```
val conf = new SparkConf()//创建spark configure
conf.set()//设置configure
val sc = new SparkContext(conf)//创建spark 运行上下文

rm_hdfs(hdfs_name,path)//删除原有的序列库

//并行执行图片特征提取任务
val fn_rdd = sc.binaryFiles(,
    initImgs_pathtask_size.toInt).map(f =>{
val fname = new Text(f._1.substring(prefix_path_hdfs))//构造记录
    的值key

val bytes = f._2.toArray()
val fcontext = new BytesWritable(bytes)//将图片的内容转化成字节
    流byte

(fname,fcontext)//以序列化方式保存图片到hdfs
}).saveAsHadoopFile(path)
```

通过上面这段序列化预处理代码，每张图片将划分到不同的分区，在不同的分区中执行图片转成单条记录的形式的任务，最后记录以 Key-Value 的形式保存到 HDFS 中。

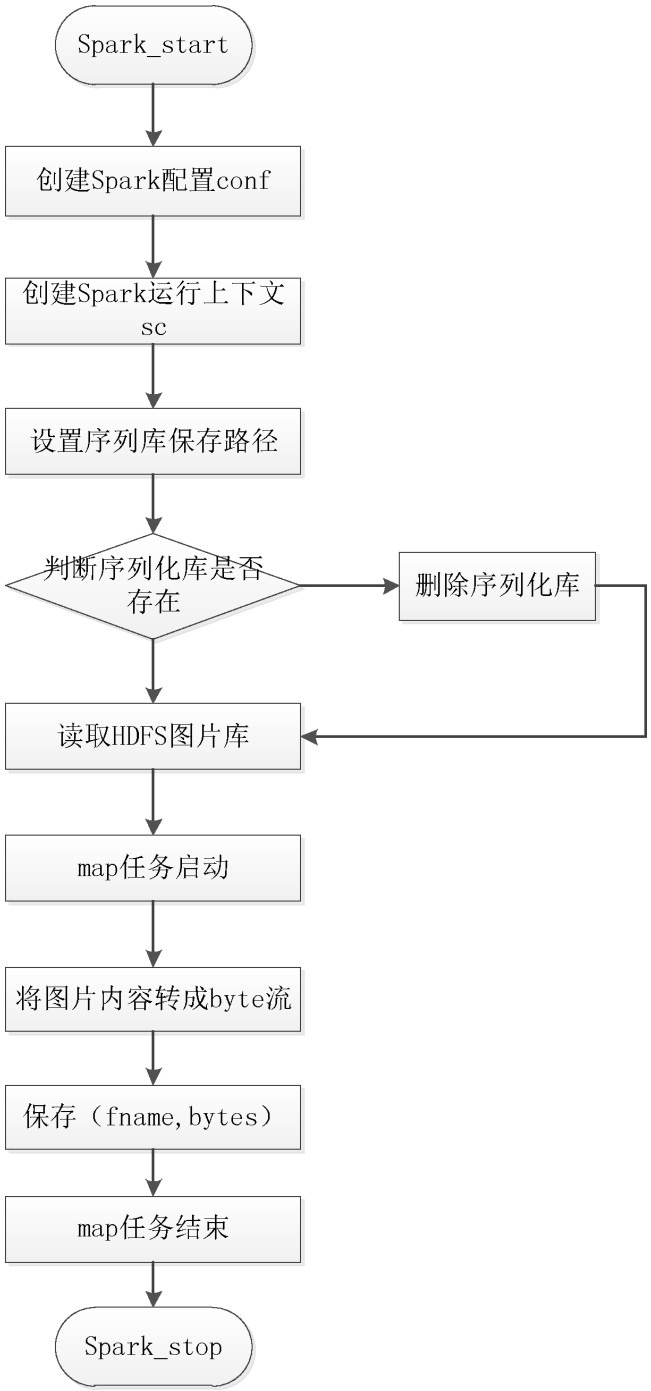

图 3.7　序列化模块流程框架图

3.2.4　存储模块

在存储层设计，本文采用 HDFS 当作系统的存储层，将图像库，图像序列化库，图像特征集，错误信息均保存到 HDFS 中。因为面对着大规模数据时，单机存储的方案基本是不太现实的，所以本文采用了分布式的存储方案。HDFS 是 Hadoop 下的产物，但是它和 Spark 的交互也十分方便。spark 下提供有丰富的

HDFS 操作 API，通过这些 API，基本上能满足开发者的需求。本文设计主要用到了 HDFS 的以下接口：

表 3.4　Spark-SIFT 系统涉及的 HDFS API

函数名	功能
sequenceFile	读取 hdfs 序列化后的图片
binaryFiles	读取 hdfs 图片库图片
saveAsHadoopFile	将图片或者是特征点集以序列化方式保存到 hdfs
FileSystem.get	获取 hdfs 文件的句柄
FileSystem.delete	删除 hdfs 文件

3.2.5　容错模块

在大数据处理中，系统的容错是一个十分重要的功能，因为处理的数据量太大，处理的时间太长，因此处理过程中发生一些错误是十分常见的情况，比如有可能处理的数据中有几个非法的数据，在 Spark-SIFT 系统处理过程中，我们经常就会碰到图片集合中有几张非法格式的图片。但是系统不能因为这个非法格式的数据而处理被中断或者是崩溃，因此大数据系统中应该有一套自己的容错机制。在 Spark-SIFT 中，系统中有一个模块会对图片格式的合法性进行判断，找出所有非法格式的图片，然后将这些文件名连同路径写到 HDFS 中非法文件保存目录下，系统会定期读取该目录，进行非法格式图片的删除。

在 Spark 中如果应用运行的过程出现了运行时的异常，应用会直接退出。因此针对这种情况，在 Spark-SIFT 系统进行特征提取时，我们对特征提取应用的每一个 stage 都加入了过滤功能，将一些非法的结果过滤掉，只传送合法的结果到下一个 stage，这样系统就不会因为发生异常而直接退出。具体实现时，本文是采用了 Scala 中的 Try 异常捕捉机制，当采用这种异常捕捉机制时，结果的返回一个重新封装的数据结构，该数据结构包含正确和异常的两个结果，换而言之，什么类型的结果都会返回，具体需要什么，我们可以根据条件进行过滤删选。图3.8显示了 Spark-SIFT 系统在一个 stage 结束之后调用容错处理操作。

3.3　系统执行与验证

在这小节中将会介绍整个 Spark-SIFT 大规模特征提取系统的编译，执行及正确性验证。

3.3.1　spark-imageLib 编译

本文设计的图像基础库需要嵌入到 spark 系统中，因此需要和 Spark 的源码一块编译，然后在 Spark 应用开发中，以依赖的形式导入 Spark-imageLib 库支持。本文使用的 Spark 版本是 2.0.0，其源代码可以在 Spark 官网或者是 github 网站下

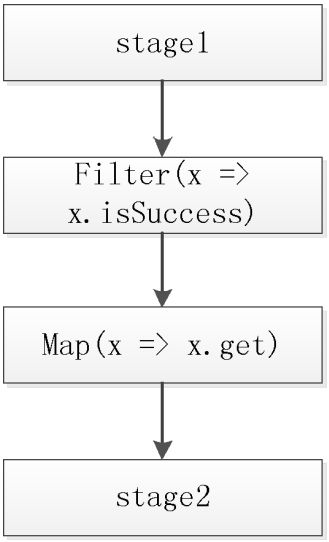

图 3.8　stage 和 stage 间的容错处理

载。Spark 源码是一个 maven 工程，maven 是一个高级的项目管理工具，可以管理项目中的所有依赖及其他配置。下面简要的介绍一下创建 Spark-imageLib 的步骤：

1. 在 Spark 源码根目录下创建一个新的 maven 工程，工程的 groupId 为 org.apache.spark，artifactId 为 spark-imageLib_2.11；
2. 修改 spark 源码根目录下的 pom.xml 文件，在 modules 配置项中加入 module 值为 imageLib 的配置项；
3. 编译源代码，编译命令为 mvn clean install，该命令会重新编译 Spark 源码，生成新的 spark 安装包，我们需要将新的 spark 安装包安装在集群上；

3.3.2　特征提取应用程序提交

在 Spark 特征提取应用程序中引用 Spark-imageLib 库依赖，编写特征提取程序，编译生成分布式特征提取应用 jar 包，然后该 jar 包提交到 Spark 运行集群上。提交命令如下：

```
spark-submit
\\--class cn.zxm.scala.sparkImageDeal.ImagesFeatureEx
\\--master spark://hadoop0:7077
spark-SIFT-1.0.SNAPSHOT.jar
```

3.3.3　正确性验证

以 280M 的图片数据集为例子，该数据集共有 1833 张图片，简单展示一下从 Spark-SIFT 系统的特征提取功能，运行提取作业结果如图3.9 所示。我们从该数据集中随机抽取了 4 张图片，分别是 ILSVRC2012_val_00021292.JPEG，ILSVRC2012_val_00021324.JPEG，ILSVRC2012_val_00021353.JPEG 及 ILSVRC2012_

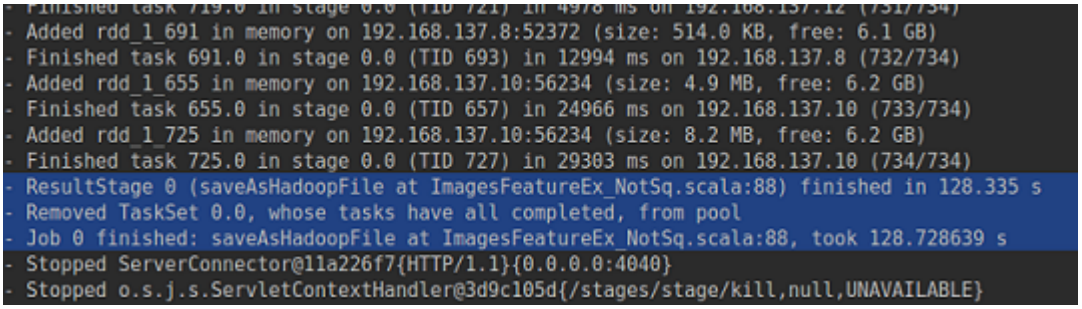

图 3.9　提取作业结果

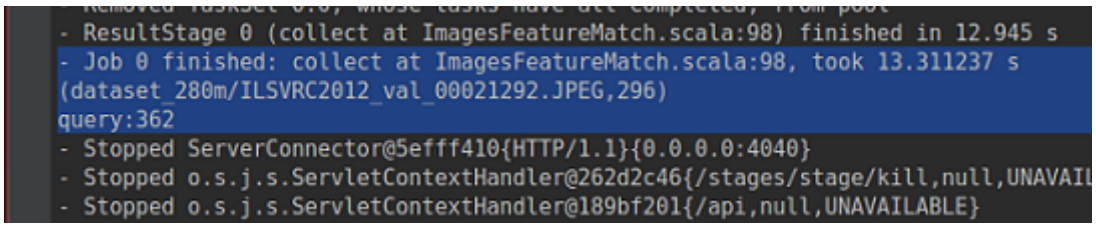

图 3.10　00021292 图片匹配结果

val_00021358.JPEG，使用这四张图片去查询匹配以验证提取的特征点的正确性，匹配程序中的匹配条件为匹配特征点数必须大于查询图片特征点的 1/2, 并且不能超过查询特征点总数。四张图片的匹配结果分别如图3.10，图3.11，图3.11，图3.13所示，其中第一张图片查询特征点总数为 362，匹配查询结果为一张，错误率为 0%，匹配点数为 296；第二张图片查询特征点总数为 1030，匹配查询结果为一张，错误率为 0%，匹配点数为 898；第三张图片查询特征点总数为 400，匹配查询结果为一张，错误率为 0%，匹配点数为 317；第四张查询特征点数为 351，匹配查询结果为 5 张，错误率为 0.27%，和原图匹配的特征点数为 318，在所有的匹配图片中是最高的。

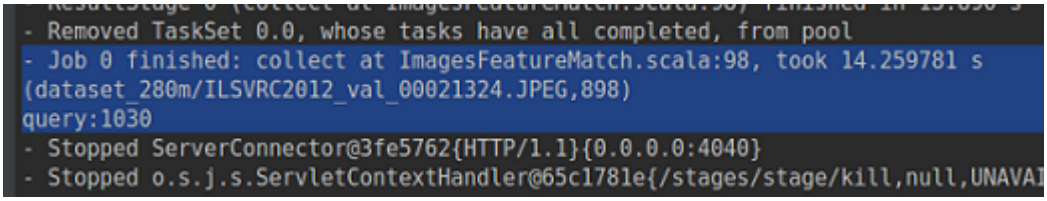

图 3.11　00021324 图片匹配结果

3.4　本章小结

本章主要介绍了大规模图像库特征提取系统 Spark-SIFT 的设计和实现，首先介绍了系统的主要框架，从软件架构的角度分析了系统的每一个模块的设计和实现，分别介绍了 Spark-imageLib，Spark-sift 算法，序列化模块，存储模块以及容错模块。最后展示了特征提取系统的编译执行及提取结果。

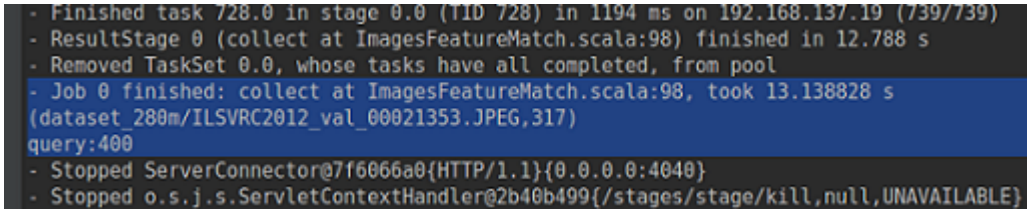

图 3.12　00021353 图片匹配结果

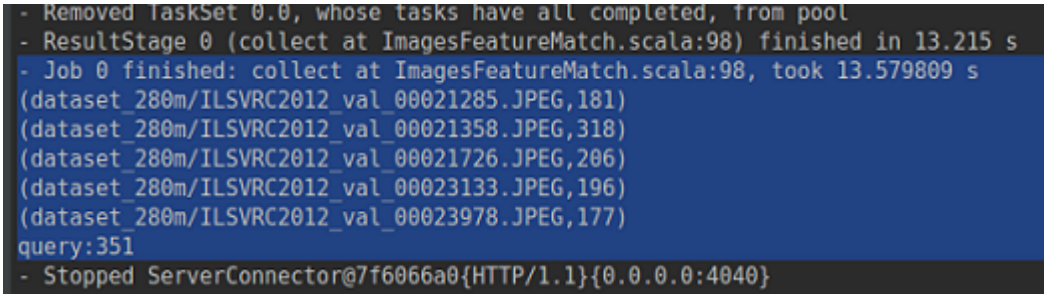

图 3.13　00021358 图片匹配结果

第四章　Spark-SIFT 系统的三种性能优化方案

在上一章节中，本文介绍了整个大规模特征提取系统 Spark-SIFT 的设计和实现，本章节主要针对 Spark-SIFT 进行性能优化。第一，针对 Spark-SIFT 系统加载大量小图片时，频繁的磁盘 IO，读写开销较大问题作出了优化，在该问题的优化上，本文设计了一种新的数据结构来描述图片，以 Key-Value 的方式表示一张图片，将图片转成单条记录的形式，然后将众多的记录合并并且序列化保存供后续操作使用，以提高读写效率；第二，由于 Spark 的任务划分及分配机制，导致 Spark-SIFT 系统在处理图片大小差异很大的数据集，会出现负载不均衡的现象，本文提出了分割式特征提取算法以解决处理过程中出现的负载不均衡问题；第三，因为分割式特征提取算法会引用 shuffle 操作，当处理的图片量很大时，会严重影响系统的性能，因此本文在分割式特征提取算法的基础上改进，提出了 shuffle-efficient 的特征提取算法，以解决 shuffle 操作带来的性能问题。

4.1　key-value 的图片描述数据结构

因为图片和图片间是分立的，整个数据集是一个个分立的文件组成，spark 在加载的时候也只能是一个个加载，如果每个文件很小，文件的数量有多，这情况下读写的效率是很低的。那么怎样才能提高加载时候的性能呢？在 spark 下处理文本信息会比较多，在处理文本信息时，spark 是把一整个文本文件加载进来，然后每一行就是一个基本的并行单元，所以能不能把一张图片看成文本文件中的一行记录，然后很多行记录组成一个大文本，就相当很多个图片合并在一个大文件中，那么一次性就可以加载很多图片。

在 spark 在处理文本文件方式的启发下，我们采用了 Key-Value 的方式来描述一张图片，Key 就是图片的文件名，Value 就是图片内容的字节流，通过这种方式，就可以用一条记录的形式描述了一张图片，然后将这些记录序列化保存到指定的 hdfs 目录下，在序列化方式上，Spark-SIFT 选用了 Hadoop 的原生序列化方式，而没有采用 Java 序列化。这是因为 Java 序列化后产生的内容容量太大，分布式环境中传输时会十分占用带宽，而 hadoop 序列化产生的内容十分简洁，容量更小，这有助于减少分布式处理时的数据传输量。通过这种将图片转换成记录的形式，Spark 在读取该目录时，就可以一次性将目录下的记录加载进来，从能提供了读写的效率。图4.1显示了整个设计的框架。

具体的实验数据将会在第五章中给出，在此先不进行数据的分析。

4.2　分割式图片特征提取算法

Spark-SIFT 在对某些数据集进行特征提取时会出现严重的负载不均衡现象。以提取 200MB 图片集合的特征为例，集合中图片的大小为 2MB 或 4MB，但其处理时间有时会比处理一个 1GB 的图片集合（图片大小在 100 到 200KB 之间）还要长。观察 Spark 的 Executors 监控页面我们发现，200MB 图片集处理时间较长

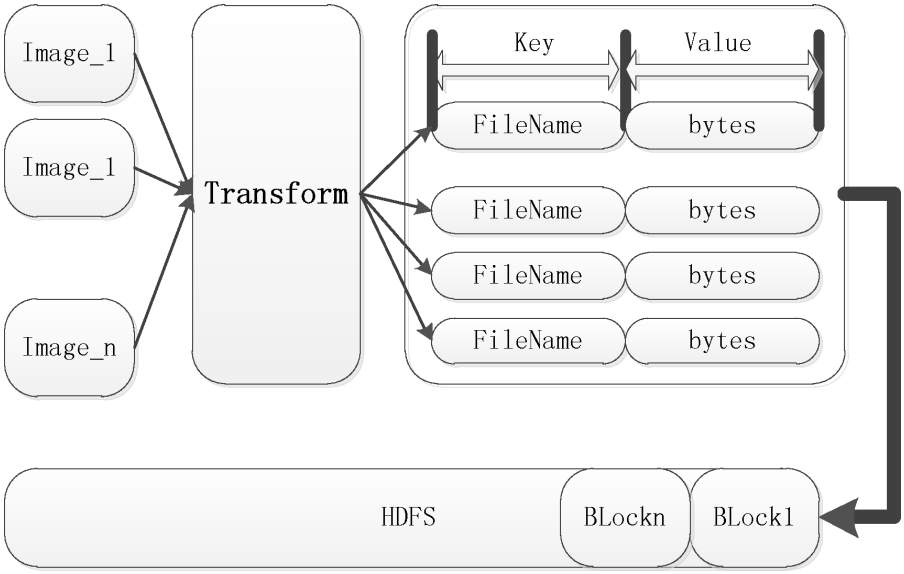

图 4.1　key-value 描述图片方式

往往是因为某个 Executor 的执行速度特别慢，而此时其他 Executors 都已经完成了分配的任务，处于空闲状态。

有两个原因引发了 Spark-SIFT 系统在处理图片差异较大数据集时出现的负载不平衡现象。首先与 SIFT 算法自身的特点有关，SIFT 算法的时间复杂度和图片大小相关，为 $O(N^5)$，并且 SIFT 算法的空间复杂度也十分大，另外 Spark 本身也是很耗内存，这样就导致运行过程中很耗内存，当内存占用过高时，CPU 的利用率无法上去，这样就导致运算速度十分慢；其次是 Spark 任务分配机制，Spark 任务分配按照大小区分，划分时仅考虑了各个任务所处理的图片总容量，而忽略了任务中图片大小的因素。那么，如果图片的大小不一样，一个任务都是大图片，而另外一个是小图片，那么大图片的运行时间会更长，但是 Spark 分配机制认为只要两个任务是 size 一样，那么两个任务运行时间基本一样。同时 Spark 分配到 Executor 上的任务，在没有失败的情况下，是不会回收的，尽管有其他空闲的 Executor，这样就会出现其他 Executors 都是空闲的，而只有一个 Executor 在运算的情况。

针对上面的问题，我们采用分割图片的方式，使图片集中各图片的大小尽可能相同，以适应 Spark 的任务分配和负载均衡策略，使各个任务的运行时间基本相同。图片分割的具体工作原理如图4.2所示：

本文将分割后得到的图片大小称为分割大小，它会影响特征提取的速度和结果的精度。理论上，分割后的图片越小，每张图片的处理速度越快，但是由于子图之间会产生边缘噪声，导致提取出的特征点存在较大的误差。通过实验，我们选择 500×500 作为基本分割大小，它既可以保证特征提取的速度足够快，又可以使得结果误差比较小，第五章将详细介绍测试结果和选择依据。

算法流程如图4.3所示，在 map 任务 1，我们将一张图片按照设置的模板大小进行划分，划分为多个子块，每个子块由图片文件名加模块下标，模块的长，模块的宽以及模块的像素内容字节数组四个元素组成。因为在 map 任务 1 结束

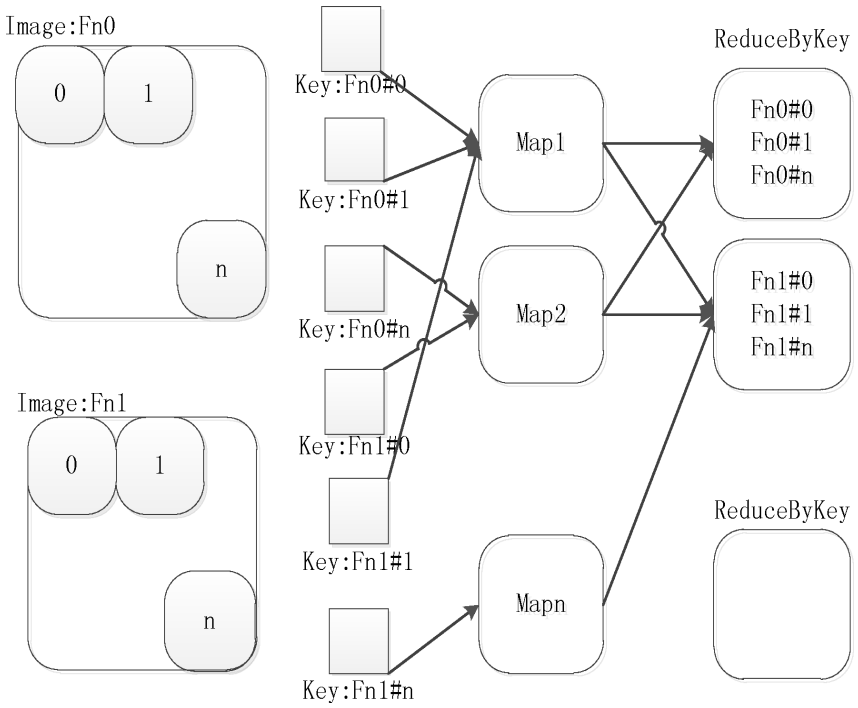

图 4.2　分割式特征提取算法

后，一张图片的子块只是位于一个独立的数组中，数组和数组间是分立的，这样并行操作只能在单个数组里面进行，这种情况并行化程度是不高的，因此在 map 任务 1 结束后，我们又启用了一个 flatMap 算子，通过 flatMap 操作，我们将数组扁平化，将每个数组的元素抽取出来，组成一个大的数组，使得并行化操作在所有图片的子块中执行，这样就可以大大提高并行化程度。当图片子块被扁平化之后，我们又重构了子块的表示方式，在 map 任务 1 中，子块的表示方式是(fname,row,col,bytes)，在 map 任务 2 中，以 (fname,(row,col,bytes)) 的方式表示一个子块，这样子块就可以表示为 key-value 的形式了，这种表示方式也为后面处理完子块后收集同一张图片的子块提供了方便；在 map 任务 3 中，进行子块图片的特征提取工作，在这个任务中，我们还进行 key 的重构工作，因为之前子块的 key 是带有下标的，但是在特征提取完之后就不需要带有下标了，因此我们将子块图片的下标去掉，那么同一张图片的子块的 key 就是一样的，最后提取的出来的特征点集以（fnkey,kpslist.tolist）的形式表示，其中 fnkey 就是子块图片的所属图片的文件名，kpslist 就是子块图片提取的特征点集合，kpslist.tolist 表示将其转化成 list 的形式；在 map 任务 3 结束后，我们启动了一个 reduceByKey 的任务，在该任务中，我们按照 key 进行 value 合并，通过这个操作，同一张图片的划分为不同子块提取的特征点集就会收集到一个数组中，数组的每一个元素对应子块提取的特征点集；在最后的一个任务 map 任务 4 中，我们将提取的特征点集进行序列化，最后保存到 HDFS 中。

子块划分算法如下所示:

算法是利用模板进行逐行扫描思想，1，2 两步获取图片被划分的块数，第 3 步进入一个嵌套循环，分别是按 rowParts 和 colParts 遍历，根据 rowOffset 和

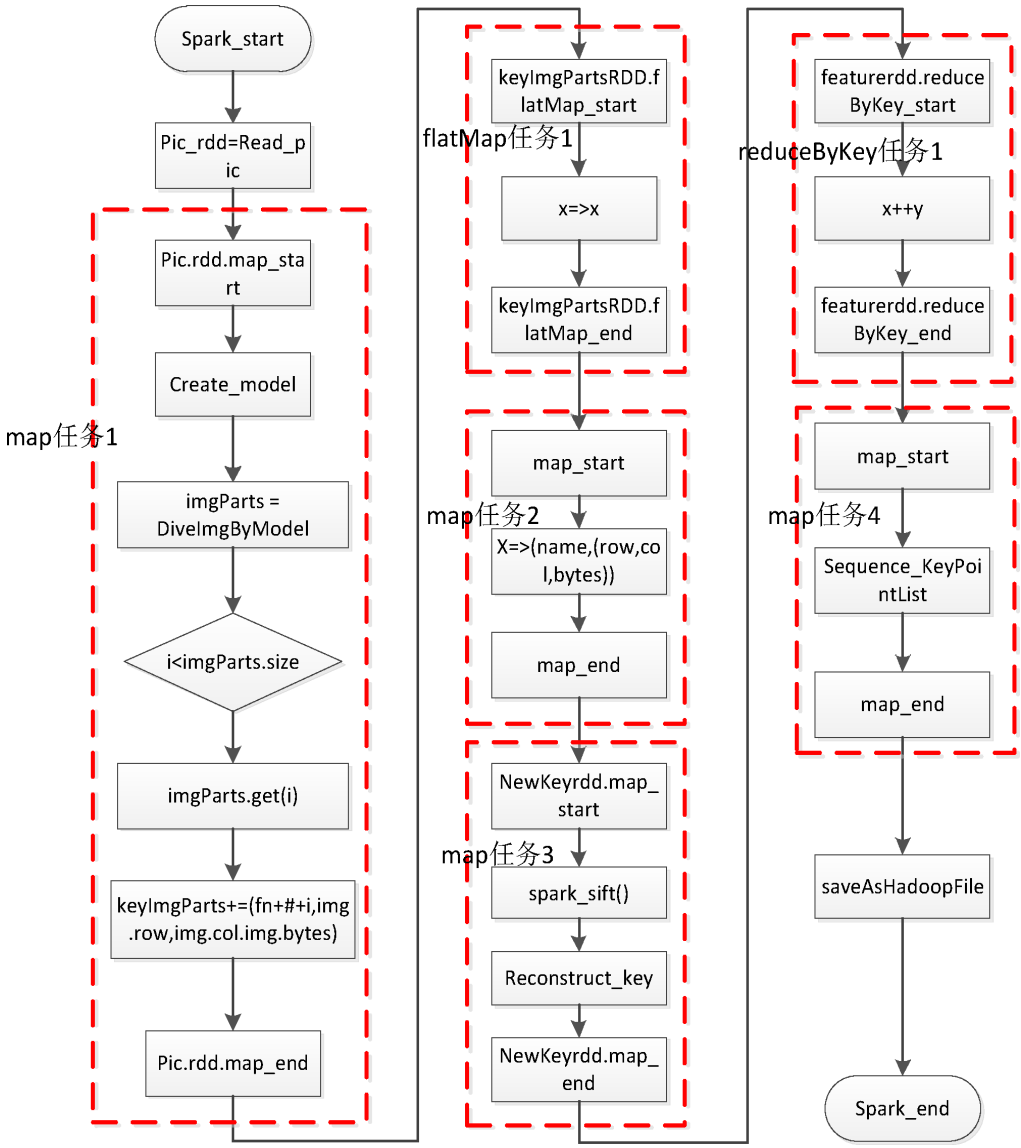

图 4.3　分割式特征提取算法流程图

colOffset 依次得到相应的子块，获取步骤为第 15 步，将每次获取到的子块添加到 imgParts 数组中，函数结束后返回 imgParts 数组。因为图片大小和模板大小不是一个整倍数的关系，所以在嵌套遍历中加入了最后一次的特殊处理，直接赋值 rowOffset 和 colOffset，即第 5 步和第 7 步。获取子块函数 getOnePart 关键伪代码如下所示：

```
public static SequenceImage GetOnePart(byte [][]pixel, int
    rowStart, int colStart, int roffset, int coffset){
    byte [][]tpixel = new byte[roffset+1][coffset+1];

    //根据行偏移和列偏移获取相应区域的像素块
    for (int i = 0; i <= roffset; i++) {
        for (int j = 0; j <= coffset; j++) {
            tpixel[i][j] = pixel[i+rowStart][j+colStart];
        }
```

算法 **4.1** 子块划分算法

输入：原始图片像素数组 orgin，划分模板 modelImg

输出：子块数组 imgParts

```
1:  rowParts ← imgrow/modelImg.row
2:  colParts ← imgcol/modelImg.col
3:  for i = 0 to rowParts do
4:      if i == rowParts − 1 then
5:          rowOffset ← row + imgrow%modelImg.row − 1
6:      else
7:          rowOffset ← row − 1
8:      end if
9:      for j = 0 to colParts do
10:         if j == colParts − 1 then
11:             colOffset ← col + imgcol%modelImg.col − 1
12:         else
13:             colOffset ← col − 1
14:         end if
15:         img ← getOnePart
16:         imgParts.add(img)
17:     end for
18: end for
19: Return imgParts
```

```
9     }
10
11    SequenceImage partimg = new SequenceImage(roffset+1,coffset
          +1,tpixel);
12
13    return partimg;//返回指定像素块
14 }
```

至此，分割式特征提取算法介绍完毕，在第五章中，我们会给出相应的实验数据分析。

4.3 Shuffle-Efficient 的特征提取算法

在上一节中，本文介绍了分割式特征提取算法，在该算法中，我们将大图片划分为子块，目的是解决 Spark-SIFT 系统提取过程中出现的负载不均衡问题。虽然分割式解决了处理过程中负载不均衡问题，但是该算法也引入了 Shuffle 操作，Shuffle 操作是 spark 处理中十分影响性能的一个操作，因此本小节中将会针对分割式提取算法中的 Shuffle 操作进行优化，以进一步提高 Spark-SIFT 系统的处理性能。

4.3.1 Shuffle 原理

Spark 在执行 Map-Reduce 操作时，需要将处理的数据集分发到各个节点上，在数据被处理完后，再从各个节点上回收结果，这就是一个具有 Shuffle 操作的过程。Shuffle 操作存在大量的磁盘 IO 和网络传输，十分损耗性能。spark 中存在两种 Shuffle 机制，分别是 Hash Based Shuffle 机制和 Sort Based Shuffle 机制。Hash Based Shuffle，正如名字所示，该 Shuffle 机制是跟 Hash 值相关的，这个 Hash 值指的是 key 的 Hash 值，Spark 会根据 key 的 Hash 值算出对应的分区，这个分区其实就是下游的一个任务，然后将上游任务的结果单独的写到一个文件中，下游的任务拉取相应的结果文件，继续任务的执行，整个过程如图4.4所示。

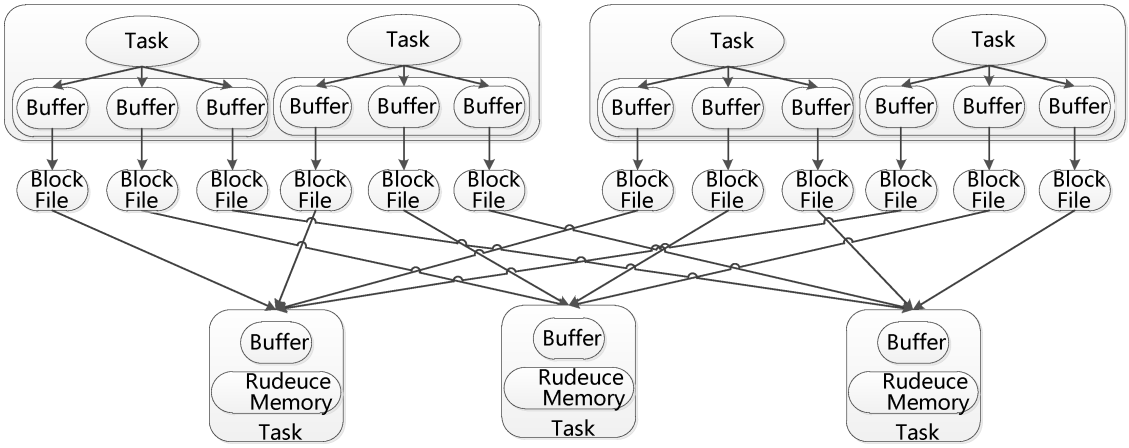

图 4.4　hash based shuffle 机制

从 Hash Based Shuffle 的运行过程就可以看出，每个上游的任务都需要给下游的任务创建一个文件，过程产生文件的总数为：tasks(上游)*tasks(下游)，在图4.4中就创建了 4*3 个过程文件，如果上游和下游的任务数很多，那么就会有大量的中间文件被创建，而每打开一个文件都会占用内存，这将消耗大量的内存，很容易就造成内存溢出；另外读写大量的文件就会存在大量的磁盘 IO 操作，这也是十分损耗性能的，并且如果同一个 key 的结果文件散落在不同的机器的分区上，又会存在网络传输的开销。

针对 Hash Based Shuffle 产生大量中间文件的问题，Spark 中又出现了一个 Sort Based Shuffle 机制，在该机制下，每个上游的任务会把结果只写到一个文件中，然后生成一个 index，下游的任务根据 index 的信息，到对应的文件上读取上游结果信息。通过这样方式，就可以大大的减少中间文件的数量，如图4.5所示，该方式下，过程文件仅为 4 个。

4.3.2 高效的分区划分

如图所示4.6在分割式算法中是存在两个 Shuffle 过程的，分别是将图片的子块分配到不同的分区上以及从不同的分区中收集子块特征点集，因为每张图片的子块在被收集之前，它们的 key 是不相等的，根据之前分割式算法分析中，它

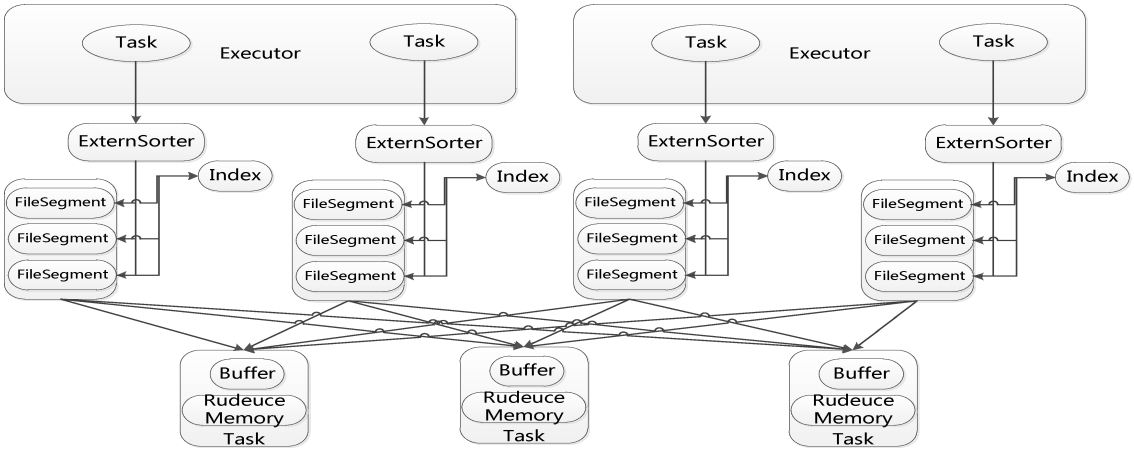

图 4.5　sort based shuffle 机制

们是用图片的文件名加上子块位于图片的下标组成的。如果按照 Spark 的默认分区策略，Spark 会按照 Key 的哈希值进行分区，那么同一张图片的子块就有可能会被分配到不同的分区上，也就有可能位于不同的机器上。所以当出现这样情况时，在收集同一张图片的子块时，就需要跨分区和跨机器收集，这显然会影响性能。因此本文提出了一种高效的分区策略，我们对子块进行分区时没有使用 Spark 的默认分区策略，而是截取了子块 key 字符串的 '#'之前的字符串，然后算这个字符串的 hash 值，根据该 hash 值对总分区数的求余值进行分区。因为同一张图片的子块它们的 key 字符串中 '#'字符之前的字符串是一样的，那么算出来的 hash 值也是一样的，所以它们的分区号也是一样的。通过这种分区方式，就可以使得同一张图片的不同子块落在同一分区上，减少了无谓的跨分区收集的网络传输开销。分区划分算法如下所示：

算法 **4.2** 高效分区策略

输入：子块 key 字符串

输出：子块分区号

```
1: newKey ← key.split("#")
2: code ← newKey.hashCode%numPartitions
3: if code < 0 then
4:         code ← code + numPartitions
5: end if
6: Return code
```

4.4　本章小结

本章主要介绍了针对 Spark-SIFT 系统的三种优化策略。首先是针对大量小文件在 spark 中加载低效问题，本文设计了一种用 key-vaule 表示图片的数据结构，将一张图片转化成一条记录，然后将多条记录合并保存，从而提高了 Spark 加载处理图片的效率；其次，本文针对 Spark-SIFT 系统在处理图片大小差异较

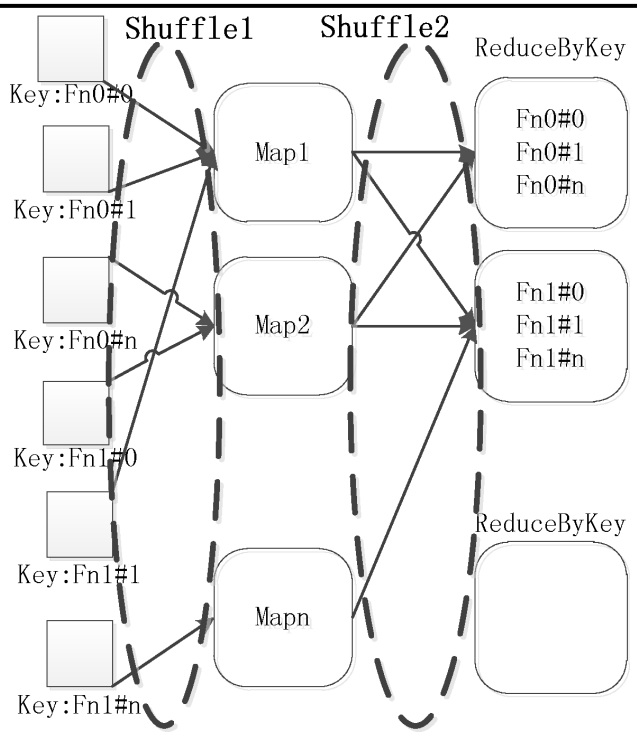

图 4.6　分割式算法的 Shuffle 操作

大的数据集时出现的负载不均衡问题，本文提出了分割式特征提取算法，将大图片划分为子块，解决了 Spark-SIFT 系统的负载不均衡问题同时也提高了系统的并行度；最后，本文针对分割式特征提取算法中引入的 Shuffle 开销，提出了 Shuffle-Efficient 的特征提取算法，使用了高效的分区策略，降低了 Shuffle 的网络开销。

第五章　Spark-SIFT 性能测试和分析

5.1　实验环境

本文构建了一个包含 7 台 PC 机的 PC 集群作为 Spark-SIFT 系统的性能测试和分析实验平台，实验平台集群的主要配置参数如下：

1. 集群总体配置 7 台 PC，其中 1 台 master，6 台 worker；Spark 的版本为 2.0.0，集群的管理模式采用 Spark 原生的 standalone 模式。
2. Master 节点配置为：处理器 AMD APU，内存为 16GB，硬盘为 1TB，操作系统为 Ubuntu 14.04；
3. Slave 节点配置为：处理器 Intel，内存 12GB，硬盘为 1TB，操作系统为 Ubuntu 14.04；

因为实验中，我们还选取了 GPU 作为对比试验，GPU 的配置为 NVIDIA GTX760，CUDA 4.0。

在测试数据集上，在测试 Key-Value 图片描述性能时，我们从 ImageNet 2012[51] 中随机选择了 8 个不同规模的图片集作为测试输入，分别是 500KB （8），70 MB（530），140 MB（1003），280 MB（1833），1 GB（8719），2 GB（15024），4 GB （28721）和 11 GB（78492），圆括号中为图片文件的数量。在测试分割式图片特征提取算法及 Shuffle-Efficent 特征提取算法时，本文采用作者平时搜集到的高清图片作为测试数据集，这是图片大都是 2M 以上，最大有 4M 多。

5.2　实验结果与分析

在实验设计上，本文首先分析 Key-Value 图片描述方式，分割式特征提取算法以及 Shuffle-Efficient 特征提取算法这三种优化策略的性能。最后本文将 Spark-SIFT 系统分别和 SIFT 的单机版本，GPU 版本对比，分析性能的差异。

5.2.1　key-value 图片描述方式

在 key-value 图片描述方式的性能分析上，本文和 binaryFile 以及 objectFile 两种方式进行对比，对比它们在加载不同的数据集大小图片所要耗费的时间，如图5.1所示，测试时，采用的并行任务数是 100 个，当加载的图片数据较小时，三种加载方式并没有什么较大的差距，比如加载 140M 时，Key-Value 图片描述方式的加载时间为 6S，binaryFile 加载方式的加载时间为 8S，objectFile 加载方式的加载时间为 8S。当加载的规模增大时，Key-Value 方式的加载性能就提升的较为明显，在加载 11G 图片数据集时，Key-Value 方式的加载时间为 39S，binaryFile 加载方式的加载时间为 102S，objectFile 加载方式的加载时间为 234S，Key-Value 图片描述方式的加载速度相对于 binaryFile 方式为 2.6 倍，相对于 objectFile 方式为 6 倍。因此在加载众多小文件时，本文设计的 Key-Value 图片描述方式大大的提高了加载的性能。

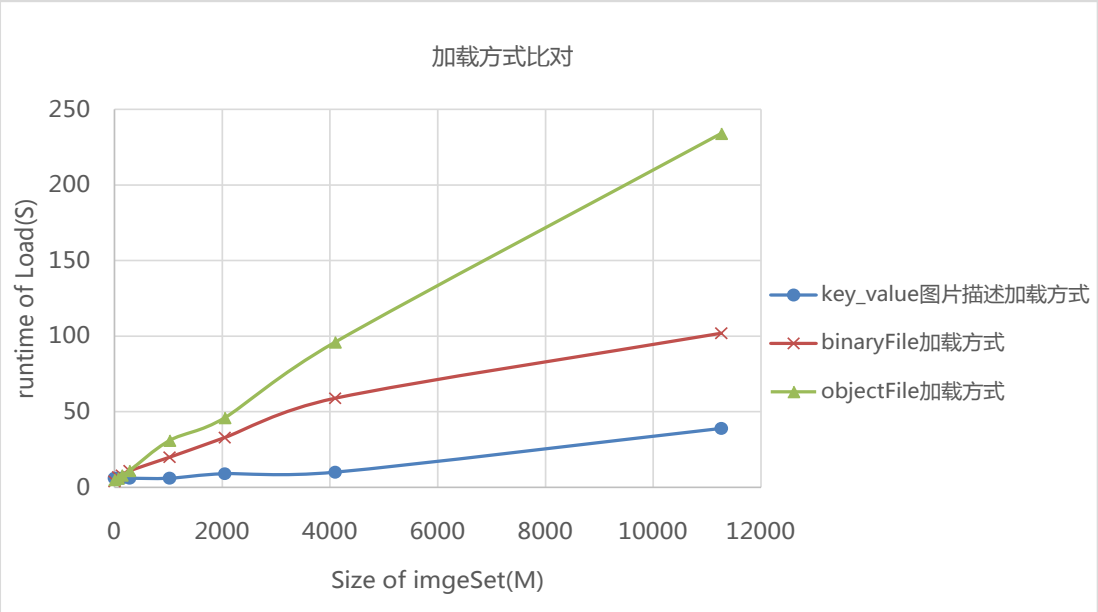

图 5.1　三种图片集合的加载性能比较

因为 Key-Value 方式是存在一个序列化预处理过程的，为了设计更加有说服力，本文也测试了序列化预处理时间占总处理时间的比例，同时如图5.2 所示，随着图片集规模的增加，序列化时间占总处理时间的比例也在不断减小。当图片规模小于 280MB 时，序列化时间占总处理时间的比例较高，超过了 10%；随着图片集的不断增大，序列化时间占总运行时间的比重会略微下降，对于 11GB 的图片集合，序列化时间为 300s，约占总处理时间的 8%。所以在处理大规模数据时，预处理并不会占用太多时间。

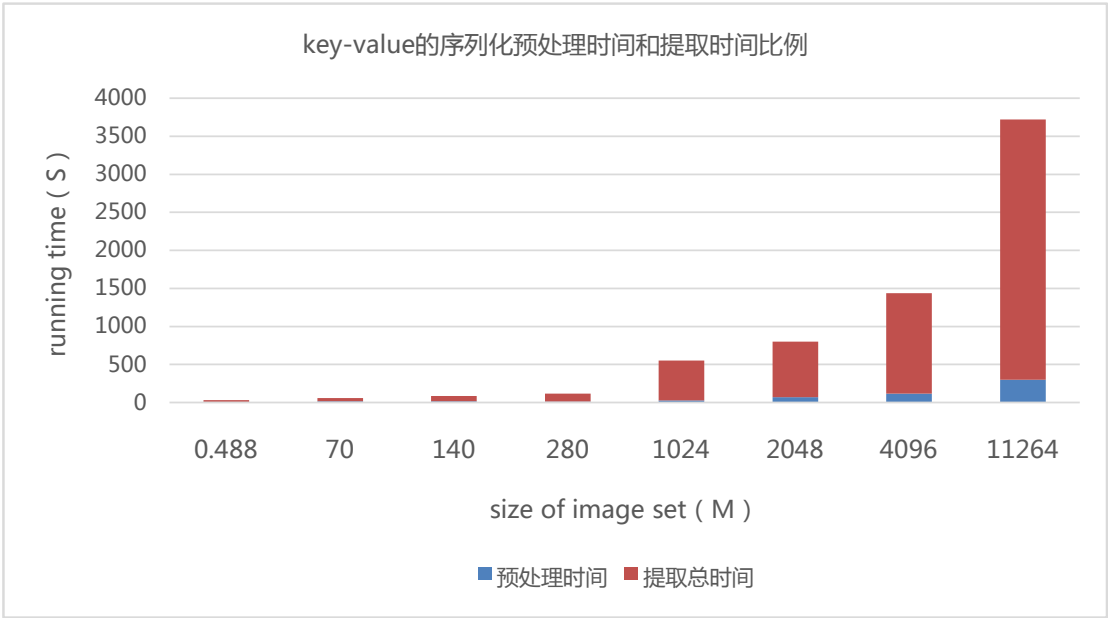

图 5.2　key-valuede 的预处理时间和提取时间比例

5.2.2 分割式特征提取算法

在测试图片分割策略的优化效果时，我们采用了总容量为 480MB 的图片集合作为输入。集合中有 280MB 的“小“ 图片，其大小在 100 到 200KB 之间，而其余 200MB 则是“ 大“图片，图片大小在 2 到 4MB 之间。当图片大小的差距增大时(例如 KB 与 GB，或 MB 与 GB)，这种优化带来的性能提升将更加明显。图5.3描述了在不同的分割大小下，Spark-SIFT 完成特征处理的时间。如图5.3所示，分割大小越小，提取的速度越快，例如在 10×10 的分割大小下，提取全部 480MB 图片特征的时间仅为 210 秒。但是如图5.4所示，分割大小越小，提取出的特征点的误差也越大，例如在 10×10 的分割大小下，误差率约为 12%。因此，综合提取速度和提取精确率两个因素，本文选取 500×500 作为分割大小。

图 5.3 不同分割大小的提取速度

5.2.3 Shuffle-Efficient 特征提取算法

在验证 Shuffle-Efficient 特征提取算法的有效性上，本文将采用高效分区策略的 Shuffle-Efficient 提取算法和采用 Hash 分区策略的分割式提取算法进行比较，在不同的测试图片数据集下对比它们在 reduceByKey 步骤上所耗费的时间，以及 Shuffle 的数据量大小。如图5.5所示，本文对高效分区策略和 Hash 分区策略分别测试了 3.2G 及 6.8G 两组图片数据集。当数据集为 3.2G 时，采用高效分区策略的子块收集时间为 169.087S，采用 Hash 分区策略的收集时间为 199.742S，时间减少了大约 30S，收集性能提高了约 15%，当测试规模增大到 6.8G 时，高效分区的收集时间为 383.532S，Hash 分区的收集时间为 546.007S，收集性能提高了 29.7%，说明图片数据集越大时，高效的分区策略越能减少收集子块时产生网络开销。

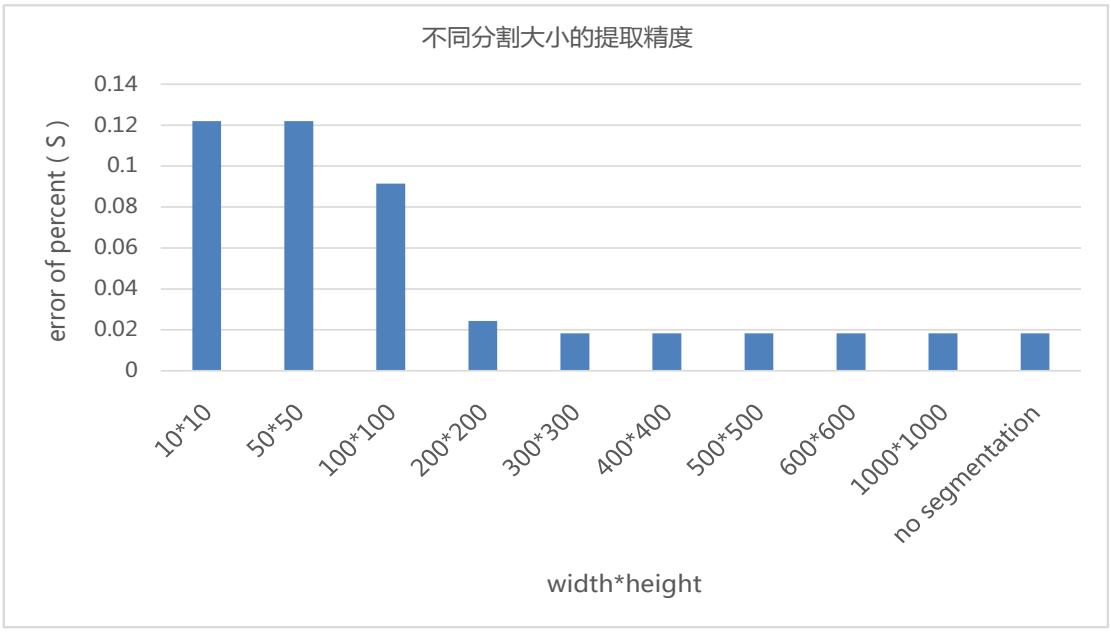

图 5.4　不同分割大小的提取精度

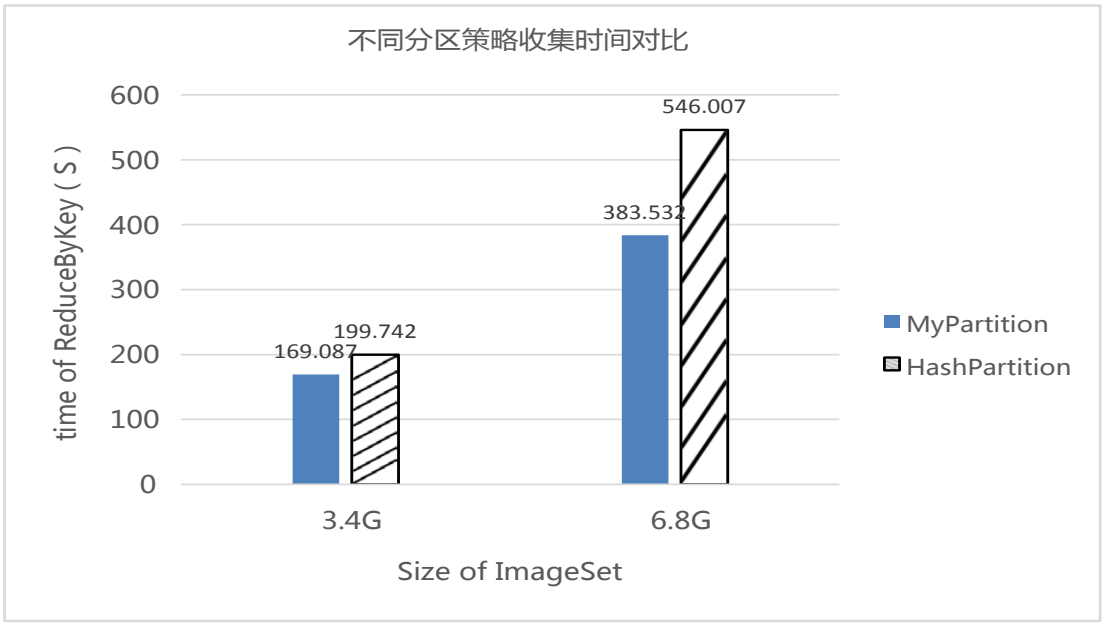

图 5.5　两种分区策略在收集子块的时间开销

5.2.4　系统综合性能对比

从图5.6可以看出，对于 500KB 的图片集，GPU 的处理时间最短，在不到 1s 的时间就完成了所有图片的特征提取；无论是否采用序列化，Spark-SIFT 的特征提取速度比单机还要慢，这是因为在图片数较少时，Spark 进行任务分配和数据收集等操作的开销占据了总运行时间中的很大一部分，此时无法体现出 Spark 的性能优势。当图片集大小为 70MB 时，Spark-SIFT 的特征速度已经优于单机版本

了，序列化方式下的提取时间为 44.8s，而单机的提取时间则为 470.71s，获得了大约 10.5 倍的性能加速。GPU 版本的处理时间为 44.98s，与序列化 Spark-SIFT 基本相同。随着图片集大小的不断增加，Spark-SIFT 在提取时间上的优势也越来越明显。例如，在 4GB 的输入规模下，Spark-SIFT 的处理时间为 1319.25s，单机版本的处理时间为 28020.03s，处理时间接近 Spark-SIFT 的 20 倍，GPU 版本的处理时间为 2671.18s，是 Spark-SIFT 的两倍。因此，对于 GB 级别以上的图片规模，Spark-SIFT 体现出较大的性能优势。

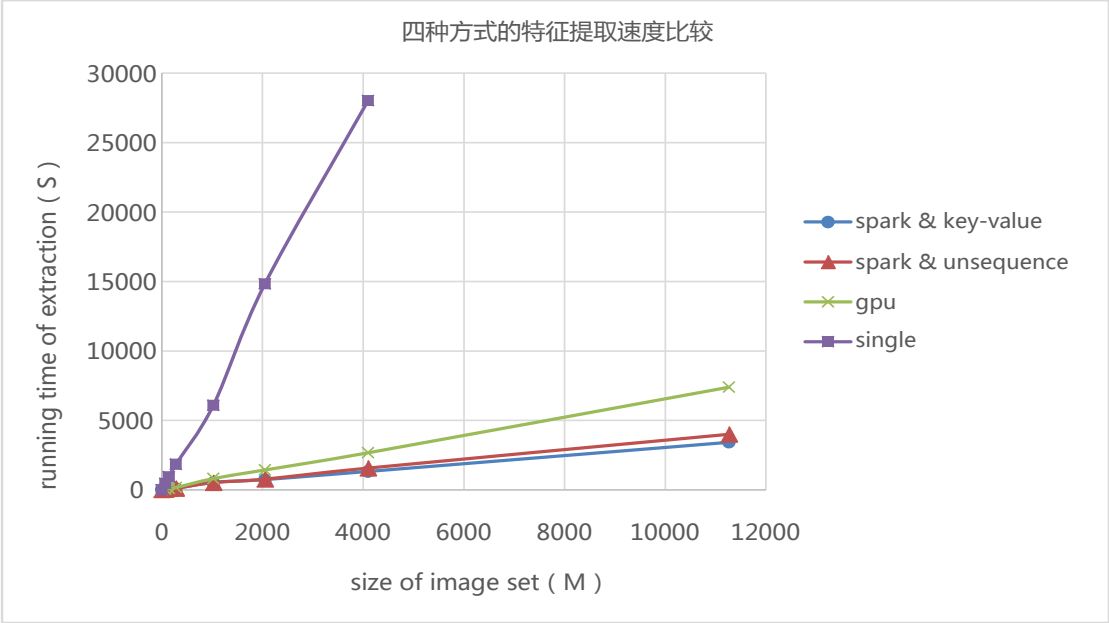

图 5.6　四种方式的特征提取速度比较

总的说来，当图片集大小为 GB 量级时，与单机版本相比，Spark-SIFT 能够带来大约 21 倍的性能加速，相对于 GPU 版本能带来大约 10 倍的性能加速，如图5.7 所示。

图5.8进一步对比了 Spark-SIFT 与 GPU 两个版本的性能，分析了访问磁盘的开销对处理时间的影响。如果没有将结果写到文件中，GPU 版本的提取速度快于 Spark-SIFT，快了大约 0.5 倍；当要将结果保存在文件中时，GPU 版本的提取速度比 Spark-SIFT 慢了约 1 倍。由此可见，访问磁盘的开销在大规模特征提取时还是相当大的。

5.3　本章小结

本章分析 Spark-SIFT 系统的综合性能以及针对其提出的三种优化方案。从实验数据来，Spark-SIFT 系统在处理 GB 级别以上的图片数据集时，速度明显快于单机的处理速度，加速比优化 GPU，从而达到了设计的预期目标。

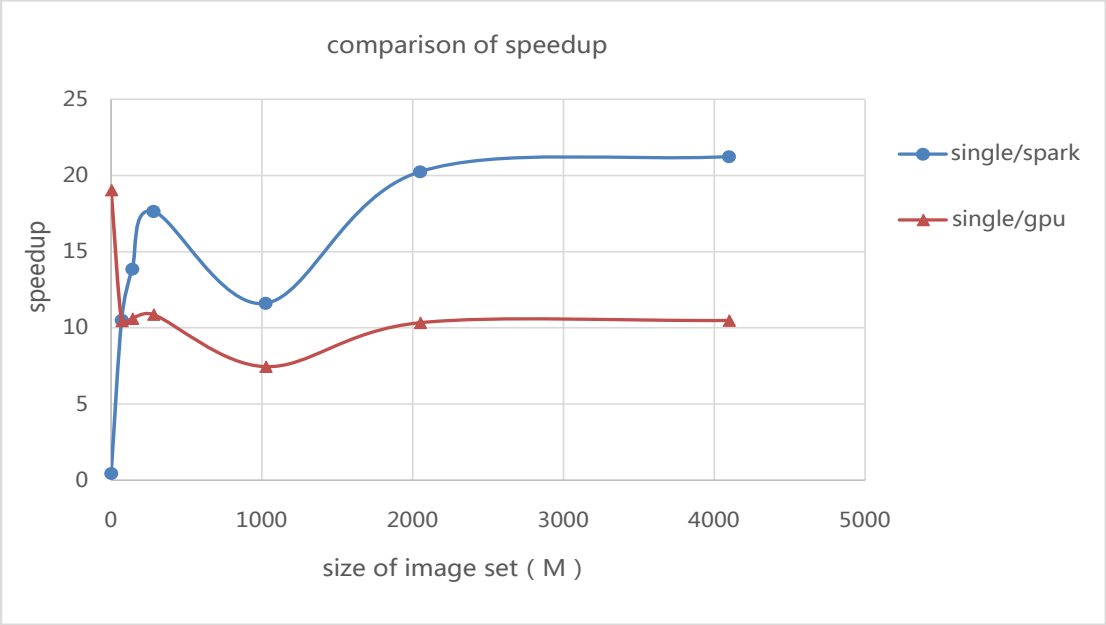

图 5.7　spark 和单机，GPU 和单机加速度比较

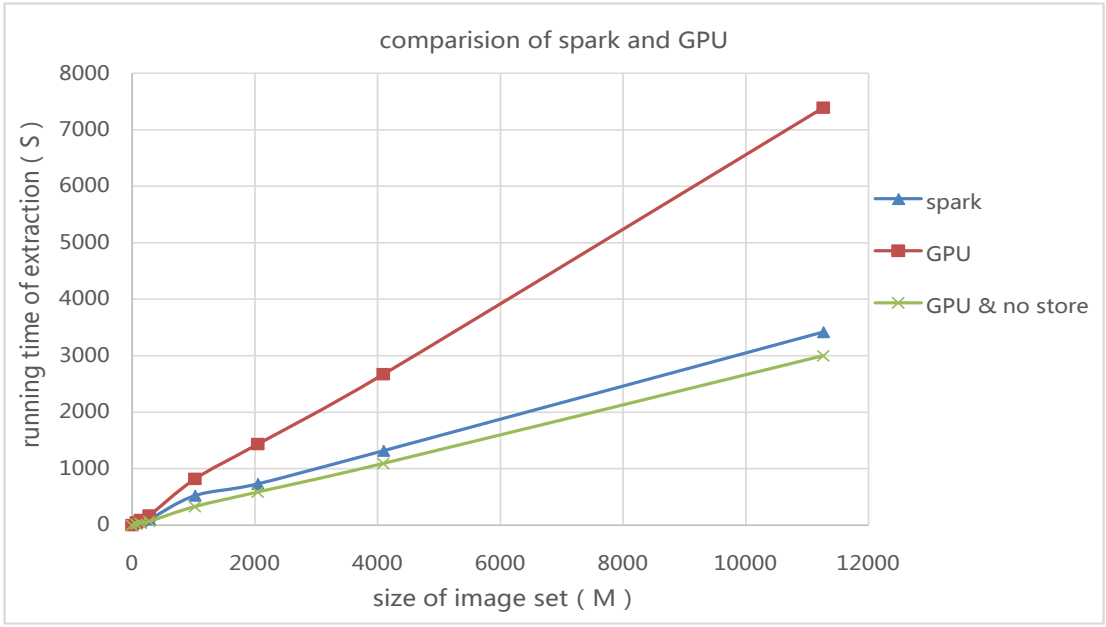

图 5.8　spark 和 GPU 两种方式比较

第六章　总结和展望

6.1 全文总结

随着大数据的不断发展，我们被越来越多的数字信息包围，如何快速的处理海量的数据逐渐成为研究的热点，比如基于图像内容的检索技术就是在该背景下发展成为研究热点，因为现在互联网上的图片越来越多，传统的基于文本的检索技术已不能应对在海量数据中准确检索的要求了。于是本文选取了图像内容检索技术中的最重要步骤特征提取技术作为研究点，研究海量图像库的快速准确特征提取技术。本文的工作可以归纳为以四点：

1. 设计了一个基于 Spark 的大规模特征提取系统 Spark-SIFT。本文选取了特征提取中具有划时代意义的算法 SIFT 算法作为 Spark 下的特征提取算法，该算法精度高，并且具有局部不变性，在目标图片被遮挡或者旋转的情况下依然可以被识别，但是该算法时间复杂度高，实时性差，于是本文根据之前研究者对 SIFT 算法优化的思路，提出了使用 Spark 进行 SIFT 算法加速的设计思路。首先，本文在 Spark 上设计了一个图像处理的基础库 Spark-imageLib，该库提供了图像的表示，图像的常用基础操作等，然后基于该 Spark-imageLib 库，本文设计了 Spark-sift 算法，最后在 Spark 应用程序中调用 Spark-sift 算法，进行大规模的图像特征提取；
2. 针对 Spark-SIFT 系统的加载处理图片低效问题，本文提出了 Key-Value 的图片描述数据结构。因为现在互联网下图片的体积普遍较小，Spark 在加载众多小图片时，会有频繁的磁盘 IO 操作，加载的效率比较低。于是，本文提出了一个以 key-value 描述图片的数据结构，将单张图片转化为一条记录，记录的 key 为文件名，value 为图片的内容字节流，然后将众多的记录序列化保存到 HDFS 中，之后 Spark 在加载时就可以一次性加载大量的记录，从而提高了 Spark 加载处理图片的效率；
3. 针对 Spark-SIFT 系统的负载不均衡问题，本文提出了分割式特征提取算法。Spark-SIFT 在进行特征提取时，我们发现 Spark 在处理图片大小差异较大的数据集时，会出现负载不均衡的现象，这是由于 Spark 在任务划分时只考虑了任务的总体积，忽略了图片尺度对任务运行时间的影响。于是本文提出了分割式的特征提取算法，将图片划分为子块，在各个子块上进行特征提取，最后归并在一起。实验数据表明分割式特征提取算法即提高了处理的并行度也解决了负载均衡的问题；
4. 针对分割式算法中存在的 Shuffle 操作，本文提出一种 Shuffle-Efficient 的特征提取算法。因为分割式提取算法虽然提高了并行度，但是也将 Shuffle 操作引进来了，Shuffle 操作会带来频繁的磁盘 IO 和网络传输开销，十分影响性能。因此，本文提出了高效的分区划分策略，对子块的 key 值的 ‘#‘字符前的字符串求哈希值，再根据哈希值对分区总数的求余值得到分区号，通过

这种方法，使得同一图片的子块散落在同一分区上，从而减少了跨分区收集子块的网络开销；

6.2 未来工作展望

随着 4K,8K 技术的发展，往后的图片的体积会越来越大，传统的技术技术肯定会面临着许多的问题，我们会对 Spark 下 4K,8K 图像处理问题展开研究。同时，我们也会在本文的基础上进行图像内容检索的研究，尝试将大规模的图像库的提取和匹配工作结合一起。最后，因为视频是由一帧帧图片组成的，因此 Spark 下如何高效进行视频处理也是一个值得深入研究的问题。

致　谢

在论文完成之际，我想对在我求学路上帮助过我，引导过我，关心过我的父母，老师，同学和朋友们致以由衷的感谢！

感谢我的导师沈立老师，沈老师在我的研究生阶段给了我很大的帮助。刚进入研究生时候，研究生和本科学习上的差异是使得我感到迷茫，沈老师在这时候帮我规划好科研方向，制定好目标，使得我逐渐的适应研究生的生活和学习。之后，沈老师不断的带领我们进行科研攻关，教会我们如何发现问题，怎样去思考问题，在组会上，沈老师总是能抓住问题的要害，分析出问题的根本，在沈老师的带领下，我的科研能力不断提高，逐渐体会到科研的快乐。进入课题之后，沈老师不断的和我讨论课题的方向，帮助我制定好课题的方向，之后沈老师严格的监督我课题的进展，耐心的指导我解决课题中的难题，在沈老师的帮助下，我的课题按照预定目标顺利完成。整个研究生阶段，我在沈老师的悉心指导下，综合素质得到了很大的提高，所以再次向沈老师表示衷心的感谢，祝愿沈老师桃李满天下！

感谢于齐师兄，于齐师兄在我进入实验室后，一直都给予我帮助。于齐师兄为人厚道，热心帮助别人，在我遇到困难的时候，师兄总是向我伸出援手，帮助我解决问题。感谢同队的张万新师兄，万新师兄在我的硕士课题上给予我很大的帮助，在我遇到困惑时，师兄总是不厌其烦的给我讲解，直到我的困惑被彻底解决。感谢钱程师兄，王博千师兄，王璐师姐，徐叶茂师兄，李宁师兄，陈静玮师姐，感谢你们对我的帮助和关怀，祝愿你们在人生道路上一帆风顺！

感谢同级同学的刘文杰，番丝江，何锡明，和你们在一起的日子里，我过得十分开心，你们的存在使得我的研究生生涯充满了欢乐！感谢李临同学，和你一块参加比赛，战斗的日子里，我留下了许多美好的回忆。祝福你们，愿你们的生活充满快乐！

感谢张诗情师妹，杨耀华师弟，感谢你们对我课题上的帮助，祝愿你们在学习科研中取得好成绩！

感谢我的父母，他们在我成长的路上付出了无比的艰辛，父母用他们辛勤的劳作换来了我现在安逸的生活。感谢我的伯父，他在人生路上给予了我很大帮助，伯父坚韧的性格一直是我学习的榜样。感谢我的大姐，大姐在我成长的路上给了我很大的帮助，无论是在学习上，工作上还是生活上，姐姐总是不断的鼓

励我，帮助我，关怀我。家人，你们无条件的爱是我强大的后盾，祝愿你们身体健康！

最后，感谢在预审，评阅及答辩过程中对我论文提出宝贵意见的专家与老师！

参考文献

[1] Internet 2011 in numbers. http://royal.pingdom.com/2012/01/17/internet-2011-inumbers.

[2] Smeulders A W M S S, Worring M. Content-Based Image Retrieval at the End of the Early Years [J]. IEEE Transactions on pattern analysis and machine intelligence. 2000, 22 (12): 1349–1380.

[3] Rui Y C S F, Huang T S. Image retrival: Current techniques, promising directions, and open issues [J]. Journal of visual communication and image representation. 1999, 10 (1): 39–62.

[4] 百度识图. http://image.baidu.com/?fr=shitu.

[5] Art and Architecture. https://www.getty.edu/research/tools/vocabularies/aat/index.html.

[6] The Google Search Engine: Commercial Search Engine founded by the Originators of PageRank. 2003. http://www.google.com/.

[7] Chunping S S L. Text clustering based on asymmetric similarity [J]. Journal of Tsinghua University. 2006, 46 (7): 1325–1328.

[8] W Niblack R B. The QBIC project : Query Image by content using color, texture and shape [J]. Storage & Retrieval for Image & Video Databases. 1993.

[9] webseekinfortech. www.webseekinfotech.com/.

[10] Sclaroff A P P. Photobook: Content-based manipulation of image databases [J]. International Journal of Computer Vision. 2006, 18 (3): 233–254.

[11] Park M R K K P. Efficient Image Retrieval Using Adaptive Segmentation of HSV Color Space [C]. In International Conference on Computational Science & its Application. 2008: 491–496.

[12] Gabillon M L C. Hue and Saturation in the RGB Color Space [C]. In International Conference on Image & Signal Processing. 2014: 203–212.

[13] Sun DingJun M Y. Summary of Texture Feature Research [J]. COMPUTER SYSTEMS & APPLICATIONS. 2010, 19 (6): 245–250.

[14] Haralick R. Statistical and structural approaches to texture [J]. Proceedings of the IEEE. 2005, 67 (5): 786–804.

[15] Vol N. Shape Feature Extraction and Classification of Food Material Using Computer Vision [J]. Transactions of the Asae. 1994, 37 (5): 1537–1545.

[16] Zhao WenBin Z Y. 角点检测技术综述 [J]. 计算机应用研究. 2006, 23 (10): 17–19.

[17] Bober F M. Robust Image Corner Detection Through Curvature Scale Space [J]. Springer Netherlands. 2003, 20 (12): 1376–1381.

[18] Okada H L. Diffusion distance for histogram comparison [J]. IEEE Computer Society. 2010, 1 (1): 246–253.

[19] Zhang L Z Z J. Feature extraction for partial discharge grayscale image based on Gray Level Co-occurrence Matrix and Local Binary Pattern [C]. 2017.

[20] Kreyszig C G H. Texture descriptors based on co-occurrence matrices [J]. Computer Vision Graphics & Image Processing. 1990, 51 (1): 70–86.

[21] Zirilli A F G M. Voronoi Diagrams – A Survey of a Fundamental Geometric Data Structure [J]. Acm Computing Surveys. 2010, 23 (3): 345–405.

[22] Hart P E. How the Hough Transform was Invented [J]. IEEE Signal Processing Magazine. 2009, 26 (6): 18–22.

[23] Schmid N D T. Human detection using oriented histograms of flow and appearance [C]. In ACM. 2004.

[24] Burge W B. Fourier Shape Descriptors [M]. Springer London, 2013.

[25] Lowe D. Object Recognition from Local Scale-Invariant Features [C]. In ICCV. 1999.

[26] Harris C. A combined corner and edge detector [J]. Proc Alvey Vision Conf. 1988, 1988 (3): 147–151.

[27] Lowe D. Distinctive Image Features from Scale-Invariant Keypoints [J]. International Journal of Computer Vision. 2004, 60 (2): 91–110.

[28] Shepard A K J. SS+-Tree: An Improved Index Structure for Similarity Searches in a High-Dimensional Feature Space [C]. In Proceedings of SPIE. 1997: 110–120.

[29] Sagiv B K. Finding and approximating top-k answers in keyword proximity search [C]. In Acm Sigact-sigmod-sigart Symposium on Principle of dataset System. 2006: 173–182.

[30] Bentley J. Multidimensional binary search trees used for associative searching [J]. Communications of the ACM. 1975, 18 (9): 509–517.

[31] Guttman A. R-trees: A dynamic index structure for sparial searching [J]. Acm Sigmod Record. 1984, 14 (2): 47–57.

[32] Satoh N K. The SR-tree: an index structure for high-dimensional nearest neighbor queries [C]. In SIGMD. 1997: 369–380.

[33] Rafiei H W C S. Locality sensitive hashing revisited: filling the gap between theory and algorithm analysis [C]. In ACM international conference on Information & Knowledge Management. 2013: 1969–1978.

[34] Das M Z C. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing [C]. In Usenix Conference on Networked Systems Design & implentation. 2012.

[35] Leung S L L. The Google file system [C]. In Nineteenth Acm Symposium on Operating Systems Priciples. 2003: 29–34.

[36] B H. SURF: Speeded Up Robust Features [C]. In CVIU. 2006: 404–417.

[37] Rabaud E R. ORB: An efficient alternative to SIFT or SURF. IEEE International Conference on Computer Vision [C]. In IEEE International Conference on Computer Vision. 2011: 2564–2571.

[38] Pajdla J M C U. Robust wide-baseline stereo from maximally stable extremal regions [J]. IEEE International Conference on Computer Vision. 2004, 22 (10): 761–767.

[39] Wiegand S H M S F. SIFT implementation and optimization for general-purpose GPU [J]. Media Culture & Society. 2007, 67 (1): 7–13.

[40] Zhu H. Image Texture Feature Extraction Based on Hadoop Cloud Platform and New ImageClass [J]. Journal of Information and Computational science. 2015, 12: 6311–6321.

[41] Sabarad A K. Color and Texture Feature Extraction using Apache Hadoop Framework [C]. In international Conference on Computing communication. 2016.

[42] Wang H. Accelerating Large-scale Image Retrieval on Heterogeneous Architecture with spark [C]. In icfm. 2015: 1023–1026.

[43] Hemachandran M S. Content Based Image Retrieval using Color and Texture [J]. Signal & Image Processing. 2012, 3: 271–273.

[44] Sparks X M B Y. MLlib: machine learning in apache spark [J]. Journal of Machine Learning Research. 2015, 17 (1): 1235–1241.

[45] Alsheikh M A, Niyato D, Lin S, et al. Mobile big data analytics using deep learning and apache spark [J]. IEEE Network. 2016, 30 (3): 22–29.

[46] Wang H, Bin W U, Liu Y. Parallel Graph Data Analysis System Based on Spark [J]. Journal of Frontiers of Computer Science & Technology. 2015.

[47] TIijima. Theory of Pattern Recognition [J]. ResearchGate. 1989.

[48] Witkin A. Scale-space filtering: A new approach to multi-scale description [C]. In IEEE International Conference on Acoustics, Speech, & Signal Processing. 2003: 150–153.

[49] Viergever B R F K. Scale-Space: Its Natural Operators and Differential Invariants [C]. In IEEE International Conference on Computer Vision. 1991: 239–255.

[50] Ghemawat J D. MapReduce: simplified data processing on large clusters [C]. In Conference on Symposium on Opearting Systems Design & Implentation. 2008: 107–113.

[51] imagenet. http://www.image-net.org/.

作者在学期间取得的学术成果

发表的学术论文

[1] Zhang xinming, Yang yaohua, Shen Li, Spark-SIFT:A Spark-Based Large-Scala Image Feature Extract System, SKG2017.

学科竞赛成果

[1] 2017 华为软件精英挑战赛武长赛区一等奖;

[2] 2016 三一重工校园 APP 设计大赛冠军;

[3] 2016 华为软件精英挑战赛武长赛区二等奖;