



来自Oracle专家的见解

Broadview
www.broadview.com.cn

Apress®

深入理解 Oracle Exadata

电子工业出版社



深入理解 Oracle Exadata

Expert Oracle Exadata

实施Oracle在存储层进行SQL处理的
突破性解决方案

Kerry Osborne
[美] **Randy Johnson** 著
Tanel Pöder



黄凯耀
张乐奕 译
张瑞

由Oracle Exadata研发部门前性能架构师Kevin Closson担任本书技术审校



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Apress®



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

由Oracle Exadata研发部门前性能架构师Kevin Closson担任本书技术审校

Tanel Pöder
[美] **Randy Johnson** 著
Kerry Osborne



张瑞
张乐奕 译
黄凯耀

apress®

内 容 简 介

本书深入地诠释了 Exadata 的各项特性,如智能扫描、混合列式存储、存储索引、智能闪存、IO 资源管理;系统地介绍了如何安装、配置和管理 Exadata;完美地阐述了 Exadata 的等待事件、性能监控和调优方法;详细地剖析了计算节点和存储节点的内部原理;全面地分享了作者们在实际项目中所获得的宝贵经验,如如何进行大数据的高效移植、Exadata 上的一些常见误区、数据库资源管理,等等。本书是实践经验的总结和升华,可读性极强,不仅有对 Exadata 深入的研究,还有对它们优雅的展现,它将带领读者进入 Exadata 的殿堂。

Expert Oracle Exadata By Kerry Osborne, Randy Johnson, Tanel Pöder, ISBN: 978-1-4302-3392-3.

Original English language edition published by Apress Media. Copyright © 2011 by Apress Media.

Simplified Chinese-language edition copyright © 2012 by Publishing House of Electronics Industry. All rights reserved.

本书中文简体版专有版权由 Apress Media 授予电子工业出版社。专有出版权受法律保护。

版权贸易合同登记号图字: 01-2011-7103

图书在版编目(CIP)数据

深入理解 Oracle Exadata / (美) 奥斯本 (Osborne, K.) 等著; 黄凯耀, 张乐奕, 张瑞译. —北京: 电子工业出版社, 2012.7

书名原文: Expert Oracle Exadata

ISBN 978-7-121-17489-6

I. ①深… II. ①奥… ②黄… ③张… ④张… III. ①关系数据库系统—数据库管理系统 IV. ①TP311.138

中国版本图书馆 CIP 数据核字 (2012) 第 143244 号

策划编辑: 张春雨

责任编辑: 贾 莉

印 刷:

装 订: 三河市鑫金马印装有限公司

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本: 787×980 1/16 印张: 37.5 字数: 900 千字

印 次: 2012 年 7 月第 1 次印刷

印 数: 3000 册 定价: 99.00 元

凡所购买电子工业出版社图书有缺损问题, 请向购买书店调换。若书店售缺, 请与本社发行部联系, 联系及邮购电话: (010) 88254888。

质量投诉请发邮件至 zlt@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线: (010) 88258888。

速 览

■ 原作者中文版序	v
■ 本书序	vii
■ 译者序 1	ix
■ 译者序 2	xi
■ 译者序 3	xv
■ 译者介绍	xvi i
■ 术语翻译约定	xix
■ 目录	xxi
■ 作者介绍	xxxi
■ 技术审校者	xxxii
■ 致谢	xxxv
■ 前言	xxxvii
■ 第 1 章 Exadata是什么	1
■ 第 2 章 卸载/智能扫描	21
■ 第 3 章 混合列式压缩	69
■ 第 4 章 存储索引	113
■ 第 5 章 Exadata智能闪存	135
■ 第 6 章 Exadata并行操作	153
■ 第 7 章 资源管理	187
■ 第 8 章 Exadata的配置	245
■ 第 9 章 Exadata的恢复	279
■ 第 10 章 Exadata等待事件	327
■ 第 11 章 理解Exadata的性能指标	353
■ 第 12 章 Exadata性能监控	387

■ 第 13 章 迁移到Exadata.....	429
■ 第 14 章 存储设计.....	477
■ 第 15 章 计算节点设计.....	505
■ 第 16 章 忘记已知.....	517
■ 附录A CellCLI和dcli.....	541
■ 附录B Exadata在线资源.....	553
■ 附录C 诊断脚本.....	555

原作者中文版序

First of all we'd like to say hello to our Chinese readers out there. A year ago at Oracle Open world 2011 we were asked if we would be interested in helping the Chinese translation team translate this book into the Chinese language. Today we're very excited to see it go to press. One of the unexpected benefits of this process was getting yet one more technical review of its contents. In a work of this size and complexity there are bound to be a few mistakes that somehow make it through the editing process. The Chinese translation team provided outstanding feedback and helped us correct and clarify where needed.

It has been a year since the English version of this book went to press and over two years since Oracle began shipping Exadata V2. We've been amazed (but not too surprised) by the speed at which Exadata has become a world-wide phenomenon. One of the unique challenges in writing a book about such new technology was dealing with rapid changes to the product itself. Fortunately Oracle appears to have spent this time stabilizing and refining the Exadata platform rather than expanding on its feature set. There have been surprisingly few visible changes to the feature set and today this book continues to be the definitive resource for learning Exadata. The examples and labs illustrated in these pages still work today and provide valuable insight to the reader. We hope you find this book helpful as you come to understand the inner workings of intelligent storage and why it is such a leap forward in database technology.

—— Kerry, Randy, Tanel

首先我们要向这本书的中国读者问好。在一年前的 Oracle Open World 2011 上，我们被问及是否愿意帮助中国的翻译小组将这本书翻译成中文，而今天，我们非常兴奋地看到中文译本即将出版。在这个过程中，我们的意外收获是这次翻译又再次为本书的内容做了一次技术审校，就本书的内容及复杂度而言，在写作过程中出现一些错漏在所难免，而中文翻译小组的出色反馈帮助我们纠正和澄清了这些错误。

至今，这本书的英文版出版已有一年，而离 Oracle 发布 Exadata V2 也已超过两年。我们被 Exadata 在全球走红的速度所震惊（但是并不太惊讶）。写作一本如此崭新技术的书籍的一大挑战就是要面对产品本身的快速变化。幸运的是，它并没有太多令人耳目一新的新增特性，时至今日，本

书仍然是学习 Exadata 的权威资料。书中展示的例子和实验仍然有效，并为读者带来了宝贵的见解。我们希望你从本书中获益，希望本书可以帮助你了解智能存储的内部工作机理，也可以帮助你理解为什么称其为数据库技术的一个飞跃。

——Kerry, Randy, Tanel

本书序

2008年9月，Oracle CEO Larry Ellison 在甲骨文全球用户大会（OOW）上宣布了软件及硬件集成一体化的数据库机——Oracle Exadata Database Machine（以下简称 Exadata）。Exadata 的推出不但震撼了业界、吸引了全球数据库专家的关注，也引起了 Oracle 数据库“粉丝”们对其技术的探究和追逐，而且也使得 Exadata 成为了网络搜索的热点 IT 词汇。

也正是由于 Exadata 在技术架构上的自我创新、功能上的丰富增强和性能上的极大优化，使得 Exadata 在市场推出 3 年多的时间里，得到了全球用户的广泛认可。目前，Exadata 的全球部署已经超过了 1000 台、用户遍及 67 个国家的 23 个行业。Exadata 数据库机已成为甲骨文 30 多年发展史中最成功的新产品。在甲骨文公司的云计算解决方案中，Exadata 作为数据库云服务器是 Oracle PaaS（Platform-as-a-Service，平台即服务）平台的基础构件，成为企业搭建云环境、构建云支撑平台的基石。

随着 Exadata 产品和技术的不断更新和广泛使用，无论是 Exadata 技术爱好者还是我们的广大用户，都非常希望有一本深入介绍 Exadata 的技术书籍，让读者不仅能从理论概念上更能从实际应用上来更好地理解和把握 Exadata 的技术机理，循序渐进地探索其内部的技术细节。

而 *Expert Oracle Exadata*（written by Kerry Osborne, Randy Johnson and Tanel Pöder）一书的问世无疑是“雪中送炭”，本书的三位作者 Kerry、Randy 和 Tanel 都是大家熟知的 Oracle 技术领域内的大师，他们不但有自己的 Oracle 技术博客，还为全球客户实施和部署 Oracle Exadata 产品，积累了丰厚的实战经验，对 Exadata 技术的精髓有切身的体验和理解。本书一经出版便成为 Amazon 网站上受人关注的书籍。

我相信作为国内读者，更希望看到本书的中文版译著。机缘巧合的是，我们国内的三位译者，也是国内 Oracle 数据库“粉丝”们熟知的技术专家——甲骨文公司的黄凯耀（Kaya）、阿里巴巴的张瑞（Jacky）、云和恩墨的张乐奕（Kamus）。

三位译者出于对 Exadata 技术的热爱和把控，以及对读者的尊重，在翻译的过程中，不是简单地对照原文完成语句的翻译，而是在译文中仔细斟酌每一句话的含义，按照中文阅读习惯加以解释，增加了阅读的流畅性、可理解性，避免了生涩的直译。特别值得一提的是，对于一些原文较为晦涩的地方，三位译者根据自己的理解增加了“译者注”，我相信这在目前大多数技术书籍的译文中并不常见。

也正如三位译者在各自的译者序中所写，从 2011 年 8 月份开始着手翻译起，三位译者之间及与原著作者之间关于本书翻译的邮件沟通，来来回回将近 500 封，同时三位译者对原文中的一些错误之处也进行了一并纠正，真可谓为本译作“锦上添花”。

我相信这本汇集了原作者、译者及其他多位大师的技术、经验和点评的专业书籍，一定能让国内有此技术爱好的读者沉浸于书中、感同身受，有所收获。我也相信读完本书，能让你感受到“欲穷千里目，更上一层楼”的境界。



Oracle 全球副总裁 喻思成

译者序 1

2011 年 7 月，我曾给中国 Oracle 用户组（ACOUG）做过一次有关 Exadata 的演讲，Exadata 的高性能在国内的 Oracle 社区中引起了不小的反响。其实，Exadata 在 2008 年底已经推出，自那时起，我们组（Oracle Real World Performance Group）就一直从事着 Exadata 上的性能测试与项目开发工作。在我的博客上也有一些相关的文章涉及 Exadata，如 2010 年 5 月前后发表的系列文章 Exadata V2 架构分析，但那都是些零碎的片断。毋庸置疑，Exadata 在国内还是陌生的，但作为 Oracle 数据库的未来发展方向，让更多的人熟悉 Exadata 无疑是一个很有意义的事情，甚至很多兄弟团队都鼓励我们组写作一本关于 Exadata 及其性能调优的书。

2011 年 8 月，一个机缘巧合地，博文视点的张春雨老师联系上我，希望我可以参与到这本书的翻译工作中。初看之下，这本书的内容非常丰富，是对 Exadata 的一个全面系统的介绍。于是我们一拍即合，这本书的翻译工作就此掀开序幕。另外两位译者是阿里巴巴的张瑞（HelloDBA）与恩墨科技的张乐奕（Kamus）。张瑞是阿里巴巴的架构师，负责数据库性能优化与应用架构改进，研究软硬件结合的数据库解决方案。张乐奕是云和恩墨的技术总监，Oracle ACE Director，也是国内知名的 Oracle 技术专家。于是，我们的中文翻译小组正式成立。

2011 年 8 月中旬，我们第一次接触到了这本书的电子版。开始的日子是忙碌的，每天工作之余，翻译上几页，这是一个锻炼人耐力的过程。出差途中，不管是在飞机上还是火车上，翻译这件事儿也帮助我打发了一些无聊时光。我学会了一个道理，积少成多，贵在坚持。不过也有难熬的时候，特别对于晦涩的章节，但最终理清作者的思路时，也会感到欢欣鼓舞，即使几小时已经倏忽间流走了。而印象最深的，是与张瑞和张乐奕对里面技术点的讨论，几百封邮件的往来帮助我们一起澄清了对原著诸多晦涩段落的理解。这是一个通力合作的过程，它是痛苦的，又是快乐的，我们只有尽情享受其中。

三位作者的文风其实各有特点。Tanel 是全球 Oracle 社区的著名人物，他的 Oracle Session Snapper 出名已久，性能优化的相关章节主要由他执笔，这两章充满睿智，里面完美地体现了 Tanel 对性能调优的理解，既有对全局的系统方法论的阐述，也有对每个性能指标含义的具体说明，还有对 SQL、存储节点的调优监控思路。翻译的时候，我总有一种心有戚戚焉的感觉。这些论述即使在非 Exadata 平台也有非常大的借鉴意义。

Randy 则具体关注 Exadata 的管理，包括了对数据库资源管理器、配置和恢复、存储节点与计算节点的详细剖析。对资源管理的详细阐述必将会刮起一阵清新之风，在消退 Oracle 资源管理器神秘感的同时，让读者也掌握了如何构建合适的资源管理模型。这些章节是对 Exadata 整体架构的

高可用性与高可配置性的极佳体现。

Tanel 和 Randy 还合作了关于如何移植的主题，这也是充满实践性的真知灼见的一章，里面提到的方案在实施中会有很好的借鉴意义。

Kerry 则专注于对 Exadata 相关特性的描述。通过大量的例子深入浅出地介绍了 Exadata 的主要特性，同时还大量地挖掘了各特性后面的细节。

由三位各有所长的作者联合执笔，终于成就了这本恢宏巨作。

我负责翻译的章节主要包括第 7、8、9、11、12 章，以及前言部分。第 7、8、9 章由 Randy 执笔，是关于系统资源管理、系统配置和恢复的内容。第 11 和 12 章由 Tanel 执笔，是关于性能优化的章节。

对翻译工作，不得不提的一点是对名词术语的翻译。坦白讲，以前阅读译著时，一个令人难受的地方就是原来很熟悉的英文术语与译者的翻译联系不起来。这每每让我有阅读原著的冲动。我们在本书的翻译过程中在尽量避免这个问题，对于大家所熟知的英文术语，我们尽量不做翻译。当不翻译真的很影响阅读的流畅性时，我们才会进行翻译，如“Grid Disk”，这个词汇在第 14 章就出现了 140 多次，不翻译会很影响阅读效果，所以虽然我们平时都直称“grid disk”，本书中我们还是把它翻译成了“网格盘”（或许“网格盘”一词也会因此流行起来呢）。当然，我们会在前面的中英文术语对照中列举出来。

2011 年 2 月，我们开始进行本书的翻译校对工作。这又是三位译者头脑碰撞的日子，我们努力清除原文理解上的每一个障碍，并积极与三位原作者就我们所发现的众多瑕疵进行了邮件讨论并加以改正，同时对于里面的重点和难点加入了独具特色的译者注。相信本书将是 Expert Oracle Exadata 全球的最新译本，同时也是独具特色的中文版本。如果读者在阅读原版时产生了疑惑，而会想起参考此中文版本，那将是我们莫大的荣幸。

看着眼前厚厚的著作，思绪万千，开始翻译的日子似很遥远，又历历在目。在这里，要感谢 Oracle Real World Performance Group，感谢杨中对翻译工作的支持，感谢帮助我做了认真细致校对工作的李昕、曲阜、董志平、陈长青、孙笑盈，感谢 Oracle 的众多同事提供的帮助和指导，特别是来自 COE（Center Of Excellence）的许向东。感谢喻总在百忙之中为本书做序。另外，还要谢谢我的妻子，她对我加班加点的翻译工作不仅毫无怨言，还从一个学习者的角度，校对了其中的一些章节。谢谢你们！

黄凯耀于深圳

2012-4-9

译者序 2

这本书的翻译计划是从 2011 年 8 月份开始的，据我所知，最早是博文视点的编辑“侠少”找到阿里巴巴的张瑞（Jacky）和甲骨文的黄凯耀（Kaya），然后 Jacky 再找到我。

实际上，我个人开始想要翻译这本 Exadata 技术书籍倒是从更早的时候就开始了，这本书在 Amazon 上的发行日期是 2011 年 8 月 9 日，其实早在 2011 年 2 月份就已经有另外一本关于 Exadata 性能的书籍（Achieving Extreme Performance with Oracle Exadata，作者全部是 Oracle 公司员工），但是论作者的知名度，仍然是本书更受人关注。最早知道这本书是从本书联合作者 Tanel Poder 的个人技术 Blog 中，那是 2011 年 3 月份，Tanel 发文说已经可以在 Apress 网站上购买新书 *Expert Oracle Exadata* 的 Alpha 版本，Tanel 是全球最受人尊重的 Oracle 技术专家之一，而一本技术书籍可以预先购买 Alpha 版本也是很稀奇的事情，再加上 Exadata 正是当今 IT 界的“当红炸子鸡”，理所当然这本书非常值得期待。在 2011 年 4 月份，我个人跟某出版社联系过，表达了如果该书可以引进中国，那么我很愿意组织人手进行翻译的工作，对方的回复是正在谈版权，之后没有消息。然后，Tanel 在 6 月份发文说，本书已经即将定稿，再之后，就是 8 月份，该书正式发售。而在正式发售的当月，博文视点就开始寻找中文版本的译者，可以说是非常迅速。而对于版权的猜测，那一定是博文视点拿到了版权，而某出版社失利了。:-D

以上的情况，让我在收到 Jacky 的邀请以后，毫不犹豫地接受了工作，无论工作如何繁忙，我都愿意让这本书的中文译者里有我的名字，这对于我而言可以说是一种荣幸。2011 年 8 月 17 日收到这本书的 PDF 电子版（当然后来又收到纸质版），从 8 月份开始，Kaya、Jacky 和我都迅速地投入了翻译的工作，在整个过程中，通过不断地沟通，我们按照每个人的经验和对各个章节的熟悉程度以及感兴趣程度，大致是均分了各个章节。我负责翻译的章节是第 1、2、4、6、13、16 章，原本我给自己定下的计划是每两周翻译一章，那么最快可以在两个月内完成翻译，再加上校稿，本来计划在 3 个月内可以完成所有的翻译，也就是如果一切顺利，这本书的中文译本应该在 2011 年年底的时候跟大家见面了。但是，计划永远是赶不上变化的，除了工作的繁忙和个人的懒惰，我们几个译者还都在其他方面出现了这样或那样的意外情况，导致整个翻译工作整体滞后。所幸，还不算太迟，我想在你们看到本书的时候，这个世界上应该还没有更新的 Exadata 书籍可以参考。所以，这本书仍然是迄今为止想要了解 Exadata、想要使用 Exadata、想要监控调整 Exadata 的最佳参考书籍。

Oracle Exadata 的举世瞩目，对整个数据库硬件/软件市场的震撼，在全球或者仅仅是中国国内的引人瞩目，乃至热销，都已经无须赘言。作为数据库从业者，也许你没有听过 Netezza，也许你没有听过 Twinfin，也许你没有听过 Hana，但是你一定听过 Exadata，这绝不仅仅是由于 Oracle 公司一贯的好战、勇于进攻、大力宣传的风格，而是 Exadata 确实具有独步天下的功能。也许我们不能说在经过最精细的调整以后，Exadata 在数据仓库领域与其他竞争对手相比一定具有绝对的优势，但是，不要忘记，在现在这个世界里，又有多少是纯粹的数据仓库系统呢？又有多少用户愿意让 OLTP 用一套系统，而数据仓库又用另外一套系统呢？这其中的数据传输开销和系统设计复杂性的开销，如果能够消减甚至避免，那么又何乐而不为呢？Exadata 正是这样的一套软硬件一体的平台，同时支持 OLTP 类型负载和数据仓库类型负载，通过 Oracle Database 11gR2 中的资源管理器来更加精细地调控硬件资源，让两种类型的负载都能获得各自需要的资源，并顺畅执行。

如果我们抛却 Exadata 在存储节点中的软件特性，它使用的各个硬件组件并不是划时代的，无论是 Infiniband 还是 Flashcache/SSD，都已经出现很久了，在企业级市场中也已被很多用户使用，但是将这些组件放在一起，并且预先调整为一个平衡的系统（没有任何一处明显的性能瓶颈），这是划时代的。Oracle 将软硬一体机的概念推广到了开放性平台上，极大地挑战了 Teradata 的市场，用开放性的硬件+开放性的操作系统+开放性的数据库软件，构造出了一个平衡的、性能超强的平台，这同样是划时代的。

好吧，前面我们提到了“抛却 Exadata 在存储节点中的软件特性”是吗？这就好比说，把皇冠上最闪亮的那颗宝石先摘下来，别闪花了我们的眼睛。现在，我们要把这颗宝石放回去了，智能扫描（Smart Scan）、存储索引（Storage Index）、混合列式压缩（Hybrid Columnar Compression），无论哪一项软件特性都足以震撼数据处理市场，而当它们结合在一起，配合上 Oracle Database 原本就具有的高性能，再配合前面说的这个平衡的硬件架构，我们就得到了足以颠覆一切固有理念的惊人性能。在 Exadata 的 POC 现场，有客户因为实在无法接受 Exadata 展示出来的飞一般的速度而怀疑 Oracle 的技术人员在造假。这在无奈的同时，无疑也是一种自豪吧。

Exadata 的出现，颠覆了一些我们既有的数据库管理理念，但是无论如何，Exadata 中运行的是 Oracle Enterprise Linux（当然也有 Solaris，不过是 x86-64 版本，至少到目前为止，Oracle 还没有计划显示会出现 SPARC 平台上的 Exadata），Linux 上运行的是 Oracle Database 11gR2，对于所有数据库技术从业者来说，之前积累的操作系统管理知识，Oracle 数据库/RAC 管理知识都仍然适用。我们需要的只是与时俱进，将 Exadata 的特有知识点加入我们以前的知识体系中。本书是最佳的入手点，因为本书中不但有 Exadata 的特性阐述，也同样有使用经验和最佳实践。要知道本书的作者都是真正的 Exadata 使用者，而本书的技术审校者（Kevin）更是 Exadata 的性能架构师（不过，Kevin 现在已经离开 Oracle 公司，加盟 EMC，去玩 Greenplum 了）。

我唯一希望的是，大家在阅读这本中文译本的时候，不至于产生去重新阅读原著的冲动（虽然，我仍然建议大家去阅读原著），因为如果那样，那只能表示我们的翻译实在是 很不适合中文读者的

理解。如果你觉得本书优秀，那么基本上可以说这是原作者的功劳，当然，我也希望你们看到我们三位译者的努力。我们在翻译完各自的章节以后，又互相审阅了其他人的章节，我们尽量斟酌每一句话的翻译，希望读起来是符合中文阅读习惯的，对于一些比较难于理解的片段（比如 Kevin 说的某些话），我们通过邮件跟作者进行了沟通以确保译文是正确体现了作者意图的，对于一些原文较为晦涩的地方，我们也根据自己的理解增加了“译者注”，我相信这也是目前大多数技术书籍的译文中并不常见的，我们甚至在想，如果译者注足够多，那么就可以出一本批注版的书籍了（:-D）。这其中，由于 Kaya 在 Exadata 中的实战经验尤为丰富，更是付出了格外的精力。你们现在看到的这本 *Expert Oracle Exadata* 的中文版，应该是全球的最新版本，因为在我们的翻译过程中，不但将本书对应英文版出版以后提交给作者的错误修订全部都更正到了本书中，而且我们还在翻译过程中发现了更多的错误，Kaya 通过邮件直接跟三位作者沟通并一一确认，最终对于确实是错误的描述也都全部做了更正。实际上，这也是本书推迟到现在才出版的原因之一。

就在今天，我重新审阅完了自己翻译的第 6 章，回顾了一下从 2011 年 8 月份开始，我们三位译者和博文视点的侠少关于翻译本书的邮件沟通，来来回回将近 300 封邮件，我相信在本书中文版最终定稿的时候，沟通邮件量一定会超过 300 封（实际上最终的沟通邮件将近 500 封）。我们扪心自问，已经尽了自己最大的努力，但是一定还会有这样或那样的不足，还望读者海涵。

最后，我要感谢我的妻子和可爱的儿子，在我工作之余的很多个深夜，我仍然在翻译此书，是我的妻子极大地包容了我，没有她的支持，没有她承担几乎全部家务和对我们年仅 1 岁多的儿子的照料，也许我的翻译进度还会拖后。谢谢你，我爱你们。感谢 Kaya、Jacky，还有博文视点的侠少，与你们关于本书翻译讨论的 500 封邮件是宝贵的财富。感谢我的大学师妹——董楠，她是《老美国志异》、《此地无人生还》、《满是镜子的房间》三本畅销书籍的译者，喜欢摇滚的朋友应该热爱这几本书籍，本书某些段落的措辞有得到她的指教。另外，我同样要感谢我所在的公司——云和恩墨的多位同事，是你们帮我承担了由于翻译工作而落下的本应属于我的工作，感谢杨廷琨（老杨同时帮助审阅了本书的第 1 章），感谢盖国强，还有帮助我审阅中文译稿的同事们——仇实、刘洋、余广宏、董禹、宋春风，译稿里面也有你们的功劳，谢谢你们。

张乐奕（Kamus）于上岛咖啡，北京
2012 年 2 月 29 日

译者序 3

2008年，Oracle在OOW上发布了与HP合作开发的Exadata V1。当时，我就对Exadata充满了好奇，很快我就在自己的博客上写了第一篇介绍Exadata的文章 *Oracle Database Machine*。现在看来，文章中很多观点都是错误和可笑的，但当时介绍Exadata的技术资料非常少，很多观点只能来自于猜测。从那时开始，我一直保持着对Exadata的关注，并在一次Oracle介绍Exadata新技术的会议上，认识了Kaya（本书的另外一位译者），这也为我们共同翻译本书埋下了伏笔。

2009年，Oracle发布了Exadata V2，它不仅采用了SUN的硬件，更是革命性地引入了Flash存储，并采用智能闪存（Exadata Smart FlashCache）技术，让Exadata同时支持DW和OLTP应用，成为真正全能型的数据库软硬件一体机。2010年2月1日，我在博客上发表了第二篇关于Exadata的技术文章《Oracle Exadata 技术浅析》，引起了大家的热烈讨论，并被广泛转载。文章中介绍了Exadata的新特性，并且在没有资料提及的情况下，提出SmartScan应该只能在特殊访问路径下（直接路径扫描）才能启用的观点。从此以后，我对Exadata着了迷，无奈除了一些官方文档以外，几乎没有其他任何资料，而我又没机会亲身操作Exadata，只能通过各种途径了解Exadata的最新信息。

2010年，我参加了在北京举办的OOW大会，第一次近距离看到了Exadata的真面目，也更加深刻体会到“Hardware and Software Engineered to Work Together”这句话的真正含义。从此，我开始致力于推动Flash存储技术在数据库领域的应用，研究软硬件结合的数据库解决方案。2011年10月举办的OTN China Tour活动上，受ACOUg的邀请，我做了《软硬件结合的数据库解决方案》主题演讲，介绍了我们在软硬件结合方面的一些尝试，并且解读了Exadata和Oracle刚推出的ODA（Oracle Database Appliance）的架构，引起了非常大的反响。

2011年底，我终于等到了亲身体验Exadata的机会，Oracle提供了一个Exadata V2 Quarter Rack供我们测试。我们根据在线交易网站的特点，专门设计了一个测试模型，并且先在高端的小型机和存储上进行测试，然后再在Exadata上运行相同的测试，以此来评估Exadata和传统主机存储之间的性能差异。Exadata的表现让我们非常惊讶，性能完全超过了小型机和存储。Exadata使用FlashCache技术，在读多写少的应用场景下，表现超乎想像，不仅预热速度快，而且性能与将数据全部放在Flash上相差无几，提供了非常好的性能价格比。不仅如此，我们还特别模拟了存储节点宕机的情况，当某个存储节点宕机时，Exadata表示毫无影响，而且性能下降非常小。通过一周的测试，我们对Exadata高性能、高可用和高度灵活的特性有了更深的认识。

作为一名Exadata技术爱好者，一直苦于找不到一本深入介绍Exadata的书籍，直到偶然一次

机会我发现了 *Expert Oracle Exadata* 这本书，本书的几位作者都是 Oracle 技术领域的大师，之前我一直订阅他们的博客，了解最新的 Oracle 前沿技术，冲着它是几位大师的合著，再看看内容简介，我知道它就是我期盼已久的那本书。当时这本书还没有完成，我找到出版社希望能在本书出版后第一时间拿到，没想到侠少不仅帮我拿到了书，同时也拿到了本书的版权，并且希望我能翻译这本书，我毫不犹豫地答应了，因为我已经期待了太长时间。但是当我读了一点又开始犹豫了，因为对于 Exadata 来说我只是个初学者，而且本书的技术含量非常高，凭我一己之力很难完成这项工作，我马上想到了两个人，一位是 Kaya(黄凯耀)，另一位是 Kamus(张乐奕)，Kaya 来自于 Oracle RealWorld Database Performance Group，专门研究 Oracle 和 Exadata 的性能优化，我相信国内没有人比他更熟悉 Exadata。Kamus 是国内知名的 Oracle 技术专家，Oracle ACE Director，他也是 Exadata 技术爱好者。事实证明，他们就是最合适的人。

虽然我以前也经常写文章，但是翻译书籍还是第一次，深深体会到翻译工作的艰辛。因为 Exadata 是业界最新的技术，翻译的过程也是学习的过程，很多技术都是第一次碰到，需要理解并准确表达出来，这对翻译者来说是个严峻的考验。我们三个人一起翻译和讨论，不知不觉就过去了半年，来来回回发了几百封邮件，往往一个句子甚至一个词，都要讨论很多遍才能确定下来，正是在这个讨论的过程中，很多技术问题都被我们搞清楚了。我们坚信翻译不是简单的文字转换，一定要自己先搞清楚，才能翻译出来给大家。正因为如此，我们在翻译的过程中，也发现了原书中的很多错误，并且 Kaya 把我们发现的错误都整理出来发给了原书的作者，所以大家读到的应该是最新版本。

如果说本书最难翻译的部分是什么，当属“Kevin 说”，因为 Kevin 说的要不是“技术哲学”，就是“技术原理”，有时候我们读过几遍之后，还不知道他在说什么，我们已经尽可能翻译得通俗易懂，但是仍然有很多不完美的地方，请大家谅解。另外，因为 Exadata 的技术非常新，我们在难以理解的地方都加上了译者注，希望可以帮助大家。本书中出现的术语，我们都采用中英文对照的方式，也有一些常用的 Oracle 术语我们选择不翻译，因为英文比中文更容易理解。

最后，我要感谢家人对我的支持，尤其是我的爱人 Emily 和宝贝 Michael，每天工作到深夜，回家后看到你们熟睡的脸庞，觉得一切付出都是值得的。还要感谢和我一起战斗过的 Kaya 和 Kamus，你们给了我太多的帮助。感谢原书的作者，正是因为你们写了这样一本伟大的著作，才有现在的中文版。最后感谢读者的支持和理解，希望你们喜欢这本书。

我翻译了本书第 3、5、10、14、15 章以及附录部分。

张 瑞

2012 年 4 月 1 日于杭州

译者介绍

黄凯耀，2006 年加入 Oracle，在 Real World Performance Group（隶属于 Oracle 公司总部数据库产品管理部门）工作，担任首席软件工程师。主要从事关键客户的现场性能测试、现实客户碰到的重大问题解决方案、Oracle 数据库的质量保证、数据库间的竞争分析等工作。特别专注于大型数据库（VLDB）在 OLTP 与 OLAP 环境下的高性能与高可扩展性的最佳实践。目前工作重点在于 Oracle Exadata 的性能测试与实施。乐于技术的总结与分享，个人技术博客为 www.os2ora.com。



张乐奕（Kamus）

云和恩墨（北京）信息技术有限公司技术总监

Oracle ACE Director

Itpub Oracle 数据库管理版/高可用版版主

ACOUG (www.acoug.org) 联合创始人

OESIG (www.oesig.org) 创始人

个人 Blog: www.dbform.com

张乐奕，云和恩墨的联合创始人之一，致力于通过不断的技术探索，帮助中国用户理解和接触新技术，推广数据库技术应用。曾先后任职于 UT 斯达康、电讯盈科、甲骨文等知名企业，担任 DBA 及技术顾问工作。现任职于云和恩墨（北京）信息技术有限公司。

具备丰富的行业经验与技术积累，对于数据库技术具有深刻的理解。热切关注 Oracle 技术和其他相关技术，对于 Oracle 数据库 RAC 以及高可用解决方案具有丰富的实践经验。长于数据库故障诊断，数据库性能调优。作为社区和网络的活跃者，在公开演讲和出版方面，多有建树。

2004 年 2 月，作为主要作者出版了《Oracle 数据库 DBA 专题技术精粹》一书。

2005 年 6 月，作为主要作者出版了《Oracle 数据库性能优化》一书。

2007 年 12 月，获 ITPUB 论坛年度原创技术文章奖，同年 3 月，被 Oracle 公司授予 Oracle ACE 称号。

2010 年 3 月，与 Eygle 联合创立 ACOUG 用户组，目前 ACOUG 是中国最活跃的 Oracle 用户组，持续进行着技术分享。

2011 年 03 月，被 Oracle 公司授予 Oracle ACE Director 称号，同年创办中国 Exadata 特别用户组。



张瑞，网名 HelloDBA，Oracle ACE，2005 年加入阿里巴巴，数据库架构师，负责数据库性能优化与应用架构改进，主导推动了阿里巴巴数据库技术的变革。同时也是 Exadata 技术爱好者，致力于推动 Flash 存储技术在数据库领域的应用，研究软硬件结合的数据库解决方案。个人有技术博客 HelloDB.net，乐于分享数据库领域的最佳实践和研究成果，并创立了 AskHelloDBA.com 专业数据库问答社区，解答各种数据库技术问题，定期举办 AskHelloDBA 数据库技术论坛。

术语翻译约定

■ A

ASM 范围安全策略 ASM-Scoped Security

■ B

半机柜 half rack
并行子进程 slave process
部分重建 partial reconstruction

■ C

CPU 量子 cpu quantum
CPU 资源短缺 CPU starvation
查询协调进程 query coordinator
出列 dequeue
串行直接路径读取 serial direct path reads
磁盘修复计时器 disk repair timer
磁盘组 disk group
存储节点 storage cell
存储节点卸载处理 cell offload processing
存储节点卸载效率 cell offload efficiency
存储节点应急过程 storage cell rescue procedure

■ D

DBM 配置器 DBM Configurator
单一客户端访问名字 Single Client Access Name

■ F

访问方法 access method

■ G

高度冗余 high redundancy
告警提醒 alert notification
骨干交换机 spine switch
管理网络 management network
过量分配 over-provisioning

■ H

互连流量 interconnect traffic
混合列式压缩 hybrid columnar compression

■ J

集群感知 cluster-aware
集群软件 clusterware
计算节点 compute node
交错分配 interleaving
节点安全策略 Cell Security
节点盘 cell disk

■ K

客户端访问网络 client access network
块内链接 intra-block chaining

■ L

类别	category
类别 IORM	Category IORM

■ N

内存并行查询	in-memory parallel query
内存并行执行	in-memory parallel execute

■ P

配置工作表	Configuration Worksheet
-------	-------------------------

■ Q

全表扫描	full scan
全机柜	full rack
全索引快速扫描	index fast full scan

■ S

SQL 监控报告	SQL Monitoring Report
闪存	flash cache
失效组	failure group
实例隔离	instance caging
使用者组	consumer group
数据仓库设备	DW Appliance
数据库范围安全策略	Database-Scoped Security
数据库间 IORM	Interdatabase IORM
数据库内部 IORM	Intradatabase IORM
双盘失效	double disk failure
私有互连	private interconnect
四分之一机柜	quarter rack

■ T

通道绑定	channel bonding
------	-----------------

■ W

外部冗余	external redundancy
完全恢复	full recovery
网格盘	grid disk

■ X

系统卷	system volume
卸载，处理能力下放	offload
行链接	row chaining
行迁移	row migration
行源	row source

■ Y

压缩单元	compression unit
压缩助手	compression advisor
验证框架	validation framework
叶子交换机	leaf switch
映射规则	mapping rule

■ Z

暂挂区	pending area
正常冗余	normal redundancy
直接路径读取	direct path read
直通	pass-through
指令	directive
重平衡	rebalance
资源计划	resource plan
字段投影	projection
自动诊断信息存储库	Automatic Diagnostic Repository

目 录

作者介绍	xxxi
技术审校者	xxxii
致谢	xxxv
前言	xxxvii
第 1 章 Exadata是什么	1
1.1 Exadata概览	2
1.2 Exadata的历史	3
1.3 不同的视角	4
1.3.1 数据仓库设备	4
1.3.2 联机事务处理机器	5
1.3.3 合并平台	5
1.4 可选配置	6
1.4.1 Exadata Database Machine X2-2	6
1.4.2 Exadata Database Machine X2-8	7
1.5 硬件组件	8
1.5.1 操作系统	9
1.5.2 数据库服务器	9
1.5.3 存储服务器	9
1.5.4 InfiniBand	10
1.5.5 闪存	10
1.5.6 磁盘	10
1.5.7 其他杂项	10
1.6 软件组件	11
1.6.1 数据库服务器软件	11
1.6.2 存储服务器软件	14
1.7 软件架构	16

1.8 总结	20
第 2 章 卸载/智能扫描	21
2.1 为何卸载如此重要	21
2.2 卸载包含了什么	25
2.2.1 字段投影	25
2.2.2 谓词过滤	30
2.2.3 存储索引	32
2.2.4 简单连接 (布隆过滤)	33
2.2.5 函数卸载	36
2.2.6 压缩/解压缩	38
2.2.7 加密/解密	40
2.2.8 虚拟列	40
2.2.9 数据挖掘模型评分	43
2.2.10 非智能扫描类型的卸载	44
2.3 智能扫描的先决条件	45
2.3.1 全扫描	45
2.3.2 直接路径读取	45
2.3.3 Exadata存储	46
2.4 无法使用智能扫描的情况	49
2.4.1 未实现的功能	49
2.4.2 转换为块运输模式	49
2.4.3 跳过某些卸载操作	50
2.5 如何验证智能扫描确实发生	50
2.5.1 10046 跟踪	52
2.5.2 性能统计 (v\$sqlstat)	53
2.5.3 卸载适用字节	55
2.5.4 SQL监控	60
2.6 参数	63
2.7 总结	67
第 3 章 混合列式压缩	69
3.1 Oracle存储概述	69
3.2 Oracle压缩机制	72
3.2.1 BASIC	72

3.2.2	OLTP	72
3.2.3	HCC	72
3.3	HCC工作机制	76
3.4	HCC性能	77
3.4.1	加载性能	77
3.4.2	查询性能	82
3.4.3	DML性能	88
3.5	预期压缩率	98
3.5.1	压缩助手	98
3.5.2	真实案例	101
3.6	限制与挑战	107
3.6.1	迁移数据到非Exadata平台	107
3.6.2	关闭串行直接路径读取	108
3.6.3	锁的问题	108
3.6.4	单行访问	109
3.7	常见的使用场景	110
3.8	总结	111
第4章	存储索引	113
4.1	结构	113
4.2	监控存储索引	114
4.2.1	数据库统计值	115
4.2.2	跟踪	116
4.2.3	总结	119
4.3	控制存储索引	119
4.3.1	_kcfis_storageidx_disabled	120
4.3.2	_kcfis_storageidx_diag_mode	120
4.3.3	_cell_storidx_mode	120
4.3.4	存储软件参数	121
4.4	行为	121
4.5	性能	122
4.5.1	为空值进行特殊优化	124
4.5.2	字段值的物理分布	125
4.6	潜在问题	127
4.6.1	不正确的结果	127

4.6.2	类型转换	128
4.6.3	分区大小	131
4.6.4	不兼容的编程技巧	131
4.7	总结	133
第 5 章	Exadata智能闪存	135
5.1	硬件	135
5.2	Cache vs. Flash Disk	136
5.2.1	使用闪存作为缓存	137
5.2.2	如何创建ESFC	142
5.2.3	控制ESFC的使用	144
5.3	监控	145
5.3.1	存储监控	145
5.3.2	数据库监控	148
5.4	性能	149
5.5	总结	151
第 6 章	Exadata并行操作	153
6.1	参数	153
6.2	存储层的并行	154
6.3	自动并行度	155
6.3.1	操作和配置	155
6.3.2	I/O基准测试	158
6.3.3	自动并行度调整小结	162
6.4	并行语句队列	162
6.4.1	老方法	162
6.4.2	新方法	163
6.4.3	控制并行队列	167
6.4.4	并行语句队列小结	175
6.5	内存并行执行	176
6.6	总结	186
第 7 章	资源管理	187
7.1	数据库资源管理器	188
7.1.1	使用者组	189
7.1.2	计划指令	193

7.1.3	资源计划	194
7.1.4	资源管理器视图	195
7.1.5	等待事件：resmgr:cpu quantum	196
7.1.6	一个DBRM的例子	197
7.1.7	测试资源计划	202
7.2	实例隔离	210
7.2.1	配置和测试实例隔离	211
7.2.2	过量分配	215
7.3	I/O 资源管理器	216
7.3.1	IORM 如何工作	218
7.3.2	Exadata管理I/O的方法	221
7.3.3	把所有的一切组合在一起	227
7.3.4	IORM 监控和指标	230
7.4	总结	244
第 8 章	Exadata的配置	245
8.1	Exadata的网络组件	245
8.1.1	管理网络	246
8.1.2	客户端访问网络	246
8.1.3	私有网络	247
8.2	关于配置过程	248
8.3	配置Exadata	250
8.3.1	第 1 步：配置工作	250
8.3.2	第 2 步：DBM 配置器	259
8.3.3	第 3 步：上传参数和部署文件	260
8.3.4	第 4 步：CheckIP (checkip.sh)	263
8.3.5	第 5 步：第一次引导Firstboot	265
8.3.6	第 6 步：准备安装介质	268
8.3.7	第 7 步：运行OneCommand	269
8.4	升级Exadata	273
8.4.1	创建一个新的RAC集群	273
8.4.2	升级现有的群集	275
8.5	总结	278
第 9 章	Exadata的恢复	279

■ 深入理解 Oracle Exadata

9.1	Exadata的诊断工具	279
9.1.1	Sun Diagnostics:sundiag.sh	280
9.1.2	健康检查HealthCheck	282
9.1.3	CellCLI	283
9.2	Exadata的备份	287
9.2.1	数据库服务器的备份	288
9.2.2	存储节点的备份	292
9.3	数据库的备份	298
9.3.1	基于磁盘的备份	298
9.3.2	基于磁带的备份	298
9.3.3	从Standby数据库上进行备份	299
9.3.4	Exadata对RMAN的优化	300
9.4	Exadata的恢复	301
9.4.1	数据库服务器的恢复	301
9.4.2	存储节点的恢复	305
9.5	总结	325
第 10 章	Exadata等待事件	327
10.1	Exadata特有的事件	327
10.1.1	节点事件	328
10.1.2	触发事件的执行计划步骤	329
10.2	用户I/O类别中的Exadata等待事件	331
10.2.1	cell smart table scan	332
10.2.2	cell smart index scan	335
10.2.3	cell single block physical read	337
10.2.4	cell multiblock physical read	339
10.2.5	cell list of blocks physical read	340
10.2.6	cell smart file creation	341
10.2.7	cell statistics gather	342
10.3	系统I/O类别中的Exadata等待事件	343
10.3.1	cell smart incremental backup	343
10.3.2	cell smart restore from backup	344
10.4	其他类别和空闲类别中的Exadata等待事件	346
10.4.1	cell smart flash unkeep	346
10.5	旧事件	347

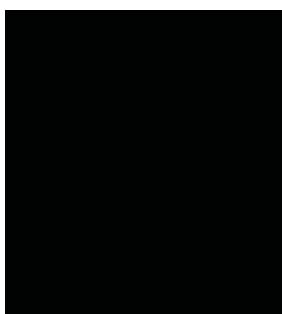
10.5.1	direct path read	347
10.5.2	enq: KO—fast object checkpoint	348
10.5.3	reliable message	349
10.6	资源管理事件	350
10.6.1	resmgr:cpu quantum	350
10.6.2	resmgr:pq queued	351
10.7	总结	352
第 11 章	理解Exadata的性能指标	353
11.1	Exadata性能指标的衡量	353
11.2	重温Exadata智能扫描的先决条件	354
11.2.1	Exadata智能扫描的性能	354
11.2.2	理解Exadata智能扫描指标和性能计数器	358
11.3	Exadata的动态性能计数器	359
11.3.1	何时及如何使用性能计数器	359
11.3.2	Exadata的性能计数器的含义和说明	363
11.3.3	Exadata的性能计数器参考	366
11.4	了解SQL语句的性能	383
11.5	总结	386
第 12 章	Exadata性能监控	387
12.1	系统方法论	387
12.2	SQL语句的响应时间监控	388
12.2.1	利用实时的SQL 监控报告对SQL语句进行监控	389
12.2.2	使用V\$SQL和V\$SQLSTATS监控SQL语句	400
12.3	监控存储节点	403
12.3.1	在存储节点利用CellCLI访问性能指标	403
12.3.2	使用Grid Control的Exadata存储服务器插件访问性能指标	404
12.3.3	使用哪些性能指标	413
12.3.4	监控Exadata 存储节点的OS性能	414
12.4	总结	427
第 13 章	迁移到Exadata	429
13.1	迁移策略	430
13.2	逻辑迁移	430
13.2.1	抽取和加载	431

■ 深入理解 Oracle Exadata

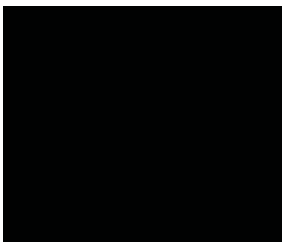
13.2.2	通过数据库链复制数据	437
13.2.3	基于同步复制的迁移	454
13.2.4	逻辑迁移小结	461
13.3	物理迁移	461
13.4	备份和恢复	462
13.4.1	全备份和恢复	462
13.4.2	增量备份	464
13.4.3	传输表空间 (和跨平台传输表空间XTTS)	465
13.4.4	物理Standby	468
13.4.5	ASM 重平衡	470
13.4.6	迁移后的任务	473
13.4.7	物理迁移小结	474
13.5	总结	475
第 14 章	存储设计	477
14.1	Exadata 磁盘架构	477
14.1.1	失效组	478
14.1.2	网格盘	480
14.1.3	存储分布	482
14.2	创建网格盘	486
14.2.1	创建网格盘	487
14.2.2	网格盘大小	488
14.2.3	创建基于闪存盘的网格盘	491
14.3	存储策略	493
14.3.1	配置选项	493
14.3.2	隔离存储节点访问	494
14.4	节点安全策略	496
14.4.1	节点安全策略的术语	496
14.4.2	节点安全策略最佳实践	497
14.4.3	配置ASM范围安全策略	497
14.4.4	配置数据库范围安全策略	499
14.4.5	删除节点安全策略	501
14.5	总结	503
第 15 章	计算节点设计	505

15.1	配置考虑.....	505
15.2	Non-RAC环境配置.....	507
15.3	RAC集群.....	511
15.4	Exadata典型配置.....	514
15.5	Exadata集群.....	515
15.6	总结.....	516
第 16 章	忘记已知.....	517
16.1	双系统记.....	517
16.2	类型负载.....	517
16.2.1	Exadata Smart Flash Cache (ESFC).....	518
16.2.2	可扩展性.....	519
16.2.3	写密集型OLTP负载.....	519
16.3	DW类型负载.....	519
16.3.1	启用智能扫描.....	520
16.3.2	阻碍智能扫描的因素.....	522
16.3.3	其他注意事项.....	533
16.4	混合负载.....	535
16.4.1	要索引还是不要？.....	535
16.4.2	优化器的局限.....	536
16.4.3	使用资源管理器.....	540
16.5	总结.....	540
附录A	CellCLI和dcli.....	541
附录B	Exadata在线资源.....	553
附录C	诊断脚本.....	555

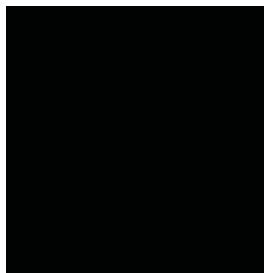
作者介绍



Kerry Osborne 自 1982 年 Oracle V2 开始就开始使用 Oracle，担任过开发人员和 DBA。在过去的几年中，他一直专注于理解 Oracle 内部机制和解决 Oracle 性能问题。他是 OakTable Network 成员（译者注：OakTable 是 Oracle 数据库领域内最高端的一个技术组织，它由一群热爱 Oracle 技术并爱刨根问底的家伙所组成，现有大约 70 位成员，他们都是最顶尖的技术专家）和 Oracle ACE Director，作为演讲者频繁出现在各种 Oracle 会议上。Kerry 还是 Enkitec 公司的联合创始人，这是一家专注于 Oracle 咨询的公司，总部在德克萨斯州的达拉斯。他的博客是 kerryosborne.oracle-guy.com。



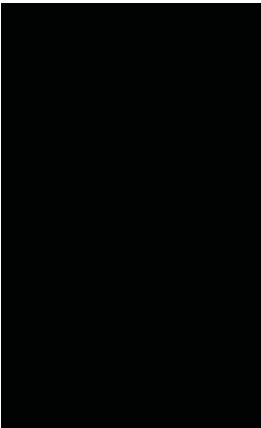
Randy Johnson 是 Enkitec 公司的首席顾问，这家公司专注于 Oracle 咨询。他从 90 年代初就开始使用 Oracle V7，已经拥有超过 18 年的 Oracle 经验。他的大部分职业生涯结合了 DBA 的工作和 UNIX 系统管理员的职责。在过去的 5 年中，他几乎将全部精力都放在 RAC 和 ASM 上。他还是 RMAN 备份和恢复的权威，曾写过一個广泛使用的自动化 RMAN 备份工具 Dixie。Randy 偶尔会在 blog.enkitec.com 上写博客。



Tanel Pödel 是世界顶尖的 Oracle 性能专家之一，曾帮助遍布五大洲二十多个国家的客户解决了复杂的问题。他擅长先进的性能优化、故障诊断和其他复杂的任务，比如以非常短的停机时间迁移超大规模数据库（VLDB）。他从 Exadata V1 开始就优化 Exadata 的性能，并且计划更深入地研究 Exadata 性能优化和故障诊断。

Tanel 是世界上第一批 OCM 之一，还是 Oracle ACE Director 以及 OakTable Network 荣誉成员。他经常在全世界各种会议上发表演讲，并在他的博客 blog.tanelpoder.com 上发表文章、脚本和工具。

技术审校者



Kevin Closson 是 EMC 公司数据计算部门的技术总监和性能架构师，自 2007 年到 2011 年，Kevin 在 Oracle 公司 Server Technology Group 的 Exadata 研发部门担任性能架构师。他 25 年的职业生涯都专注于高端 SMP、UNIX 集群和 Linux 平台上的系统和软件的性能工程与架构、竞争性基准测试、技术支持和软件开发工作。他在 Oracle 工作之前，曾在 HP/PolyServe、Veritas 和 IBM/Sequent 工作，专注于提升 Oracle 服务器和支持平台的可用性和可伸缩性。因为他在 Oracle 所从事的工作，使得他在 SMP/NUMA 锁和数据库缓存机制方面获得了两项美国专利（6389513 和 6480918）。Kevin 还维护着一个非常受欢迎的技术博客 kevinclosson.wordpress.com。

致 谢

事情总是这样，最亲近的人总会为你的一时冲动付出代价。因此，我想借此机会感谢在为此项目奋斗期间我的家人对我的支持。写作本书的工作量远远超过了我的预期，但最令我惊奇的是，一年来的每一天，有一种危机感时刻笼罩着我。我开玩笑地说，所有事情都可以以段落来衡量。我可以和我的妻子到外边共赴晚宴，或者我可以写另外两个段落。（事实上，我不可能写得那么快，但你知道我要表达的意思。）因此，我要谢谢爱妻 Jill 与孩子，在闷头写作期间，他们只能远远望向我，还要承受我带给大家的一次次扫兴。

Kerry Osborne

我想要感谢我的孩子们，Brandon 和 Charis，当我在厨房的桌子上写作此书时，他们度过了很多个漫长的夜晚和周末。他们是我灵感的源泉。我的女儿 Charis，一个真正的天才作家，是如此的富有创意和胆量。我的儿子 Brandon，是家庭中才华横溢的科学家和战略思考家，他从不放弃。看着他们长大成人，这教会我创造力和毅力的价值所在，特别当任务似乎不可以逾越时。我爱你们，孩子们。

我的父母、家人和朋友们，我想对你们说一声谢谢，感谢一直以来的鼓励和支持。当我需要你们的时候，你们永远站在我身边。在这里我不一一列出每个人的名字，但你们会明白我的心意的。

最后，我想特别感谢 Kerry 邀请我与他一起完成这个旅程。几年前，如果你告诉我某一天我打算写一本书，我都会笑出声来。但是天才的 Kerry 让我从舒适区走了出来。除了是一位导师，这些年来他已经成为了一个值得信赖的朋友。

Randy Johnson

这是我合作撰写的第一本书，我崇尚以数据、算法和计算机去解决问题，借此机会，对在我的职业生涯中帮助过我、支持我想法的朋友们表示感谢。谢谢你们的教导、帮助和建议！我想要特别感谢我的父母，当我是一个小孩子时，他们就强烈支持我在技术、电子和计算机方面的兴趣。另外，非常感谢我的妻子 Janika，虽然她不是一名计算机迷，但她理解有时连续 24 小时盯着计算机屏幕是再正常不过的行为了。

Tanel Pöder

前言

感谢你购买本书。我们已经为此努力工作了很长时间，希望本书对你即将开始的 Exadata 之旅有所帮助。我们试图以一种系统的方式介绍 Exadata 的各个主题，并遵循从一般讨论到具体技术细节的思路。本书中一部分内容从宏观上描绘了 Exadata 是如何工作的，另一部分内容则是非常技术性的细节描述，你可能会发现若手边有一台 Exadata 系统以对书中所描述的技术进行试验，会更容易理解这些细节。注意，我们使用了很多隐含参数和特性以演示软件的各部分是如何工作的，请不要把它们当做管理生产系统的推荐方式。记住我们访问的系统允许我们随意玩转而不必担心由此造成的一切后果，这为研究 Exadata 如何工作带来了很多好处，除了这个特别的访问权限，我们也得到了 Oracle 公司内外部很多人的大力支持，我们对此心怀感激。

本书面向的读者

本书面对有经验的 Oracle 从业人员。除非与 Exadata 平台有关，否则我们不会试图解释 Oracle 的方方面面，这意味着我们对读者的知识背景有所假定。我们并不期望你是 Oracle 性能调优的专家，但确实期望你能够熟练使用 SQL，并对 Oracle 的体系结构有基本的了解。

写作本书的由来

在 2010 年春天，Enkitec 买了一台四分之一机柜配置的 Exadata。我们把它放在位于 Dallas 办公室的一个很小的计算机室里。我们没有正规机房的那种装配式地板或者任何花哨的东西，但这个空间确实拥有独立的空调系统。事实上，让 Oracle 答应我们的购买请求比你想像的要难得多。他们的很多客户都急切地想购买，而且对于这个新生婴儿他们爱护有加。我们没有安放 Exadata 的一流数据中心，甚至在交付机器之前，电源设施都必须加以改造。无论如何，机器交付不久，在与 Jonathan Gennick 数次交谈后，Randy 和我决定为 Apress 写作本书。在那时并没有太多文档可供参考，我们只好不断打扰可以找到的任何一个了解 Exadata 的人。

在 2010 年春天的 Hotsos 论坛（Hotsos Symposium）上，Kevin Closson 和 Dan Norris 非常热心地回答了我们的很多问题。过后 Kevin 联系了我，并自荐为本书正式的技术评审员。于是，Randy 和我在整个夏天和初秋都在辛苦地学习，以尽可能掌握所有的东西。

在 2010 年 9 月的 Oracle Open World 大会上，我幸运地碰到了 Tanel，我们说起了一个 Exadata

客户，Tanel 刚好帮助这个客户做了一些移植工作。好事成双，最终他同意加入作者的行列。在这次 Open World 大会上，Oracle 宣布了新的 X2 型号的机器，我们刚刚起步，却已经落后于技术的发展了。

2011 年 1 月，X2 平台开始出现在客户现场。Enkitec 再一次决定对此技术进行投资，我们很自豪地拥有了一台四分之一机柜配置的 Exadata X2-2。事实上，我们决定利用 X2 的组件把原有的 V2 四分之一机柜升级到半机柜。这看起来既有利于学习升级，又可以了解把两个不同版本的组件混合在一起是否有问题（事实上没有问题）。这是一个重要的里程碑。

一切都在变化

就如大多数新软件一样，Exadata 自 2008 年底推出以来迅速发展。它带来的变化包含了很多重要的新功能。事实上，此项目的难点之一在于跟上这些变化。由于写作本书期间软件行为的变化，一些章节历经了几次修订。在本书中我们试图涵盖的最后一个版本组合是数据库 11.2.0.2 bundle patch 6 和 cellsrv 11.2.2.3.2。注意在过去两年间发布了很多补丁，同时有多种数据库版本、补丁级别和 cellsrv 版本的可能组合。因此，很有可能你观察到的行为与我们所记录的并不一致。我们欢迎你的反馈，并会很高兴地解决你所发现的不一致的问题。事实上，这本书是 Apress 的 Alpha 计划的一部分，它允许读者下载这本书的早期草稿。在我们写作和编辑此书的过程中，此计划的参与者们提供了很多反馈。我们内心充满感激，更惊讶于你们所提供的如此之多的详尽的细节信息。

对志愿编辑们的感谢

在此项目上，我们得到了一些人的大力支持。让正式技术评审员在书中插入评论是有那么一点古怪。在这种情况下，谁去审核评审员的文字呢？幸运的是，在项目的前期，Arup Nanda 就担任起了编辑志愿者的角色。因此，除了作者们相互评审各自的章节，Kevin 评审所有的章节，Arup 会阅读所有内容并进行评论，这其中也包含 Kevin 的评论。另外，Oak Table Network 的很多成员在此过程中对各章节也给予了反馈，尤其是 Frits Hoogland 和 Peter Bach 提供了宝贵的意见。

当本书被加入到 Apress 的 Alpha 计划后，我们得到了一批新的评审员。通过这种格式出版的章节的早期版本得到了很多人的反馈。感谢给我们提出问题和在某些议题上帮助我们理清思路的所有人。特别是 Oracle 的 Tyler Muth 对此项目表现出了极大的兴趣，给我们提供了非常详细的反馈。他同时给我们和 Oracle 内部的其他资源架起了桥梁，如 Sue Lee，她对资源管理一章进行了非常细致的评审。

最后，我想感谢 Enkitec 的技术团队。有许多人帮助我们在此道路上前行，在 Randy 和我工作于此项目上时（而非我们真正的工作），他们帮助我们拾起落下的工作。帮助我们的人太多，我无法一一列举每个人的名字。如果你在 Enkitec 工作，并且在过去的几年里从事有关 Exadata 的工作，你已经为本书做出了贡献。我还想特别感谢 Tim Fox，虽然他忙于很多事情，包括他自己的出书项

目，但他还是为我们绘制了大量的示意图。我们也要特别感谢此项目的另一个主要贡献者 Andy Colvin，他在几方面对我们帮助很大，首先，他主要负责维护我们的测试环境，包括对平台的升级和打补丁工作，从而我们可以在第一时间测试最新的特性和软件变化带来的影响，其次，在 Randy 和我忙于写作期间，他在实施 Exadata 的客户现场帮忙保持整个团队的战斗力，再次，他帮助我们弄清楚各种特性是如何工作的，特别是关于安装、配置和连接到外部系统这些主题，如果没有他，这个项目会很难完成。

本书的作者

本书有三位作者，如果算上 Kevin，那就是四位了。这是我们四人通力合作的结果。但为了方便工作划分，我们都赞成各自负责一部分章节。开始时 Randy 和我启动这个项目，不久 Tanel 加入进来（因此在任务分配方面，他的工作会稍微轻一些，不过他是这个团队中非常有价值的一部分，他帮助我们对那些不具体分配给他的领域进行了研究）。以下是本书工作的具体分工。

Kerry: 第 1~6 章、第 10 章、第 16 章。

Randy: 第 7~9 章、第 14~15 章、第 13 章的一半。

Tanel: 第 11~12 章、第 13 章的另一半。

Kevin: 以“Kevin 说”标志的段落。

联机资源

在这本书中，我们使用了一些脚本。如果脚本比较短，或者我们觉得脚本本身值得关注，我们会在书上包含它的内容。如果脚本太长，或者不是非常有意思，我们可能会选择不包含脚本的内容。你可以在网站 www.ExpertOracleExadata.com 上找到我们在本书中使用的所有脚本的源代码。附录 C 包含了所有诊断脚本的列表，以及对它们使用目的的一个简要说明。

关于“Kevin 说”的一点说明

Kevin Closson 是本书的主要技术评审员，Kevin 是 Oracle 的 SAGE 项目的首席性能架构师，SAGE 其实就是 Exadata 的前身，因此，他不但对 Exadata 如何工作，而且对 Exadata 应该如何工作和为什么如此工作都有非常深入的了解。作为技术评审员，他的职责是对我们所写的东西进行评审，并验证它的正确性。一般的工作流程是：一个作者提交某一章节的草稿版本，Kevin 对其进行评审并附以相关的评论。随着工作的展开，我们发现把 Kevin 的一些评论加入到本书中或许是一个好主意，这能够给大家提供一个了解此过程的独特视角。Kevin 的评论非常特别，是言简意赅的典范。在此项目过程中，我发现我自己会多次回去查阅那些短小的评论和邮件，随着对某个主题的进一步熟悉，我发现它们表达了更多的含义。因此我建议你也采用类似的方式，当阅读一个章节时同时阅读他的评论，在完成这一章节时，请尝试重新阅读他的评论，我想在第二遍时，你会发现你读懂了

更多的东西。

我们如何测试

当我们开始此项目时，数据库的版本是 11.2.0.1。因此一些章节的初始测试环境采用的就是这个版本的数据库，还有在存储节点上的各种级别的补丁。当 11.2.0.2 发布后，我们回去进行重新测试。我们会努力指出显著不同的地方，但有些部分是在 11.2.0.2 发布之后才开始编写的，因此对于这些部分，我们可能不会提及其与 11.2.0.1 行为上的不同。我们使用 V2 和 X2 硬件组件的组合进行测试工作。除了 X2 的速度更快之外，它们之间基本上没有什么区别。

数据库用户和表

你会发现本书的一些例子用到了几个数据库表。Tanel 使用了一个叫做 T 的表，如下所示：

```
SYS@SANDBOX1> @table_stats
```

```
Owner : TANEL
```

```
Table : T
```

Name	Null?	Type
OWNER		VARCHAR2(30)
NAME		VARCHAR2(30)
TYPE		VARCHAR2(12)
LINE		NUMBER
TEXT		VARCHAR2(4000)
ROWNUM		NUMBER

Table Statistics

TABLE_NAME	: T
LAST_ANALYZED	: 10-APR-2011 13:28:55
DEGREE	: 1
PARTITIONED	: NO
NUM_ROWS	: 62985999
CHAIN_CNT	: 0
BLOCKS	: 1085255
EMPTY_BLOCKS	: 0
AVG_SPACE	: 0
AVG_ROW_LEN	: 104
MONITORING	: YES
SAMPLE_SIZE	: 62985999

```
-----
```

Column Statistics						
Name	Analyzed	NDV	Density	# Nulls	# Buckets	Sample
OWNER	04/10/2011	21	.047619	0	1	62985999
NAME	04/10/2011	5417	.000185	0	1	62985999
TYPE	04/10/2011	9	.111111	0	1	62985999
LINE	04/10/2011	23548	.000042	0	1	62985999
TEXT	04/10/2011	303648	.000003	0	1	62985999
ROWNUM	04/10/2011	100	.010000	0	1	62985999

我使用 SKEW 表的几个不同的变体，最常使用的是 SKEW3，如下所示：

```
SYS@SANDBOX1> @table_stats
Owner : KSO
Table : SKEW3
Name                                     Null?   Type
-----
PK_COL                                  NUMBER
COL1                                    NUMBER
COL2                                    VARCHAR2(30)
COL3                                    DATE
COL4                                    VARCHAR2(1)
NULL_COL                                VARCHAR2(10)

Table Statistics
TABLE_NAME          : SKEW3
LAST_ANALYZED       : 10-JAN-2011 19:49:00
DEGREE              : 1
PARTITIONED         : NO
NUM_ROWS            : 384000048
CHAIN_CNT           : 0
BLOCKS              : 1958654
EMPTY_BLOCKS        : 0
AVG_SPACE           : 0
AVG_ROW_LEN         : 33
MONITORING          : YES
SAMPLE_SIZE         : 384000048
-----
```

Column Statistics						
Name	Analyzed	NDV	Density	# Nulls	# Buckets	Sample
PK_COL	01/10/2011	31909888	.000000	12	1	384000036
COL1	01/10/2011	902848	.000001	4	1	384000044
COL2	01/10/2011	2	.500000	12	1	384000036
COL3	01/10/2011	1000512	.000001	12	1	384000036
COL4	01/10/2011	3	.333333	12	1	384000036
NULL_COL	01/10/2011	1	1.000000	383999049	1	999

这些详细的信息对于理解示例应该不是必需的，但如果你对这些表有什么疑问，这里的信息可供参考。当然我们也使用了其他的表，这些只是我们最常使用的。

祝你好运

我们的 Exadata 探索之旅如同一场狂欢的盛宴。希望你能如我们一样，尽情享受这个探索历程，也希望这本书能够提供一个构建你自己知识框架的平台。我觉得我们只是刚刚掀开了 Exadata 所展现的无尽可能性的面纱的一角。开始你的研究之旅吧，祝你好运，请随时在网站 www.ExpertOracleExadata.com 上向我们提问，并和我们分享你的发现。

第 1 章

Exadata 是什么？

毫无疑问你应该已经很了解 Exadata 是什么了，否则你现在手上也不会拿着这本书。在我们看来，它是一个把硬件和软件根据合理的配置整合在一起的 Oracle 数据库（在本书编写时是 11gR2 版本）平台。Exadata 数据库机器包含了存储子系统，在存储层上运行着研发的新软件，这使得研发人员可以做一些在其他平台上无法完成的事情。实际上，Exadata 一开始是以一个存储系统形式诞生的，如果你跟参与研发此产品的人交谈，你经常会听到他们称存储组件为 Exadata 或者是 SAGE（Storage Appliance for Grid Environments，网格环境存储设备），这是该产品研发项目的代码名称。

Exadata 原本设计用来解决超大型数据库所存在的普遍性能瓶颈，也就是无法（在可接受时间内）从磁盘存储系统向数据库服务器传输足够大量的数据。Oracle 一直以来都提供了非常快速的数据访问方式，主要是通过非常聪明的缓存技术，但是当数据库容量开始扩大到无法高效使用缓存技术的时候，Oracle 开始寻找一种解决方案，可以有效解决存在于存储层和数据库层之间的瓶颈。他们想出的解决方案是融合了硬件和软件的组合拳。如果你仔细想一下，也可以知道有两种方法可以减小这个瓶颈，第一种是让传输管道更大，虽然有很多组件参与其中，不过你可以简单地认为 Infiniband 就是这个更大的管道；第二种方法是减少需要传输的数据量，这是通过智能扫描（Smart Scan）来完成的。这套组合非常成功地解决了问题，但是不要误会，通过智能扫描实现的后者（第二种方法）才是真正的制胜秘诀。

■ Kevin 说：如果前提是尽可能地减少对于 Oracle 数据库核心功能的修改的话，作者们对于减轻数据仓库/商业智能系统的存储和服务器之间瓶颈的方法论描述是正确的，不过实际上，从一个纯粹的计算机科学角度看，存储和服务器之间数据流问题的解决方案中还应该包括划分多个数据库实例读取数据的方法，也就是“shared-nothing”这种 MPP 方法。虽然这也值得一提，不过还是不把时间浪费在讨论 Oracle 并未采用的方案上了。

在本介绍性章节中，我们将介绍组成 Exadata 的组件，包括硬件和软件，我们会讨论它们是如何协同工作的（也就是整体架构），我们还会谈到数据库服务器是如何跟存储服务器交互的，在这一点上 Exadata 跟其他平台非常不同，所以我们会花较多时间来讨论。另外，我们还会提及一些历史背景。在本章结束时，读者应该对于这些部件如何组合在一起并协同工作有一个全面的概念，而本书后续章节中会对本章搭建出来的框架中的知识点做更详细的描述。

■ Kevin 说：在我看来，数据仓库/商业智能从业者，在 Oracle 环境中，如果对 Exadata 感兴趣，那么在了解 Exadata 其他方面之前，最应该先了解的是存储节点卸载 (Offload) 处理，而所有的其他技术都只是为了支持存储节点的卸载处理。比如，过早过多地关注 Exadata Infiniband 组件，就并非是深刻理解整个技术的最好方式，换句话说，我们在观察细节之前最好先欣赏全局。当我做 Exadata 主题培训的时候，我都是从存储节点卸载处理开始，讲解下面四个概念。

存储节点卸载处理 (Cell Offload Processing)：该工作由存储服务器完成，否则就必须在数据库服务器上执行。它包括的功能有智能扫描、数据文件初始化、RMAN offload 和混合列式压缩解压 (在没有内存并行查询的场景中)。

智能扫描 (Smart Scan)：这是存储节点卸载处理中与提高数据仓库/商业智能查询性能的最有关系的操作，智能扫描是将数据文件初始化、字段投影、存储索引消减以及 HCC 解压等操作卸载到存储节点的方法。

全表扫描或者全索引快速扫描 (Full Scan or Index Fast Full Scan)：为了触发智能扫描，查询优化器选择的必要的访问方法。

直接路径读取 (Direct Path Read)：智能扫描需要的缓存模型，智能扫描的数据流是无法缓存在 SGA 缓冲池中的。直接路径读取可以串行也可以并行，缓冲在进程的 PGA (heap) 中。

1.1 Exadata 概览

一图胜千言。图 1-1 展示了组成 Exadata 数据库机器的各部分的轮廓。

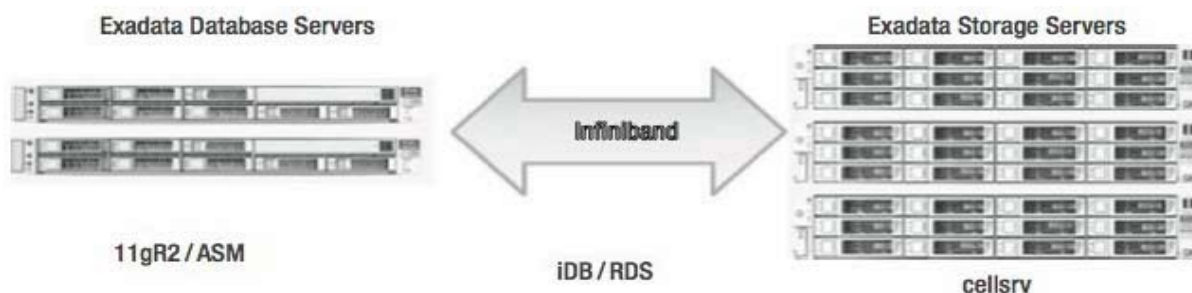


图 1-1 Exadata 组件概览

当考虑 Exadata 的时候，如果在思想上将整个系统划分为两个部分会更好理解，即存储层和数

数据库层。两层之间用 Infiniband 网络来连接，Infiniband 提供了低延迟、高带宽的光纤通信链路，也提供了链路上的冗余和联结（bonding）。数据库层由多个 Sun 服务器组成，运行着 Oracle 11gR2 软件，虽然 RAC 实际上并不是必须的，但是这些服务器通常还是配置成一个或多个 RAC 集群。数据库服务器使用 ASM 来管理存储，即使数据库并不是配置成 RAC，ASM 也是必须的。存储层同样由多个 Sun 服务器构成，每个存储服务器都包含 12 块磁盘，并且运行着 Oracle 存储服务器软件（称之为 cellsrv）。两层之间的通信通过 iDB 协议完成，这是一种使用 Infiniband 来实现的网络协议，iDB 用来将请求和请求的元数据（比如说查询谓词）传输到 cellsrv 中，在某些情况下，cellsrv 可以利用请求元数据来访问需要的数据，而后再将结果传输回数据库层。当 cellsrv 如此运作的时候，就被称为智能扫描，通常因此将会极大减少需要传输回数据库层的数据量。当无法使用智能扫描的时候，cellsrv 就会返回整个 Oracle 数据块。iDB 使用的是 RDS 协议，这是一种低延迟的协议，跳过了内核调用，在 Infiniband 网络中通过远程直接内存访问（RDMA）来完成进程间通信。

1.2 Exadata 的历史

自 2008 年首次发布以来，Exadata 已经经历了好几次重大变革。实际上，编写本书的一大难点就在于我们要时刻跟上 Exadata 平台的改变。以下简要回顾产品的沿袭和与时俱进的变迁。

■ Kevin 说：我想分享一些历史。在 Exadata 之前，我们有一个 SAGE——网格环境存储设备，这大概可以称为 V0。实际上直到为了在 2008 年的 Open World 时大张旗鼓地跟惠普一起缔造一个超强联合品牌，拉里埃里森将之命名为 Exadata 之前的一星期，它还一直被称为 SAGE。虽然 SAGE 的第一个实现版本完全采用了 HP 的硬件，不过 Oracle 始终没有认为这个平台一定要运行于 HP 硬件之上，在 Oracle 收购 Sun 后，Exadata 平台更是完全地采用了 Sun 的硬件。另一方面，Oracle 也没有认为数据库主机一定要运行于 Linux 操作系统之上，事实上就在收购 Sun 成为事实之前，把 iDB 移植到 HP-UX 安腾平台的开发工作已经接近尾声了。要说 SAGE 的诞生，其实可以追溯到更早的时候。

V1：2008 年末发布的第一款 Exadata，被称为 V1，融合了 HP 的硬件和 Oracle 的软件。整体架构跟现在的 X2-2 版本相似，除了 Flash Cache 部分（这是在 V2 版本中新加入的）。Exadata V1 市场定位在纯粹的数据仓库平台。该产品广受关注但是并没有大面积部署，由于硬件过热带来的麻烦也让人头疼，总是听到“在机柜上都可以煎鸡蛋了”这样的说法。很多原先 V1 的客户后来都替换成了 V2。

V2：2009 年的 Open World 上发布了 Exadata 第 2 版本。这个版本是 Sun 和 Oracle 合作的产品。当

发布该产品时，Oracle 已经在尝试收购 Sun 微系统公司。很多组件都升级到更大更快的版本，不过最大的区别是 V2 采用了大容量的固态存储。存储节点升级为使用了 384G 的 Exadata 智能闪存，而软件也同样升级，以充分发挥这些新缓存的效能。这让 Oracle 可以将产品定位于不仅仅是数据仓库的平台，由此打开了一个更大的市场。

X2：2010 年的 Open World 上发布了 Exadata 第 3 版本，被称为 X2。实际上 X2 有两个独立的版本。X2-2 依循的是 V2 的基本架构，但是数据库服务器升级到了 8 台双 CPU 服务器，CPU 也升级到了 6 核，之前的 V2 是 4 核。另外一个 X2 版本是 X2-8，它没有使用小型的 1U 数据库服务器，而是引入了更大的拥有 8×8 核 CPU 核、1T 内存的更大的数据库服务器，由于具有更多的 CPU 和更大的内存，X2-8 的市场定位是为大型 OLTP 系统或者混合负载系统提供更强劲、更稳定的平台。

1.3 不同的视角

前面已经介绍了我们是如何看待 Exadata 的，但是，正如大家熟知的盲人摸象的故事里说的那样，关于 Exadata 的本质存在着很多互相矛盾的说法。本节我们来看看这几种描述。

1.3.1 数据仓库设备

有时候 Exadata 会被称为“数据仓库设备”(DW Appliance)，虽然 Oracle 一直在尝试不让 Exadata 仅定位于这一类型，但这可能是对 Exadata 最真实的描述了。一个硬件和软件紧密集成的盒子，Oracle 期望客户不用太多修改就能直接运行，这实际上跟我们对数据仓库设备的普遍理解是一致的。不过，Oracle 数据库的特性决定了 Exadata 是可以高度自由配置的，在这一点上又跟通常的数据仓库设备不太一样，要知道通常这些设备是没有太多东西可以调节的。不过，在数据仓库设备和 Exadata 之间仍然有一些共通的特性。

出众的性能：通常我们认为 Exadata 和数据仓库设备之间最大的共同点就是，它们都是为数据仓库类型查询做了优化的。

快速部署：数据仓库设备和 Exadata 都能非常快速地进行部署，因为 Exadata 是预先配置好的，通常能在到货以后一周内运行起来，这与一个普通的 Oracle 集群数据库通常需要花费好几周来部署形成了鲜明对比。

扩展性：两种平台都具备高扩展性的架构，Exadata 的升级是可以分步骤完成的，从一个半机柜配置升级到全机柜，将会在升级数据库服务器计算能力的同时，同步升级磁盘的整体吞吐能力。

整体成本降低：这听上去可能有些稀奇，因为很多人认为 Exadata 最大的问题就是价格昂贵，但是实际上无论是数据仓库设备还是 Exadata，都会在多种应用环境中降低整体成本。在 Exadata 中，这要部分归功于为了提供指定处理能力所必须的 Oracle 数据库许可证的减少。我们已经看到过多次，一些公司在多种硬件平台上测试评估 Oracle 应用，最后还是由于成本减少的原因，选择了 Exadata 平台。

高可用性：大多数数据仓库设备都提供了或多或少的高可用架构，由于 Exadata 运行的是普通的 Oracle 11g 软件，因此所有的 Oracle 提供的高可用特性都是可用的，并且在硬件设计上也避免了任何一处单点故障。

预先配置：当 Exadata 交付到你的数据中心时，一位 Sun 工程师将会安排时间，协助你进行初始化配置。这包括了确认整个机柜的连线正确以及所有的预期功能都正常。但是像大多数数据仓库设备一样，各部件之间的整合工作是早就完成了的，因此无须大量的研究和测试。

有限的标准配置：大多数数据仓库设备只提供非常有限的几种配置（如小、中、大），Exadata 也是如此。目前只有 4 种可能的配置，这与支持力度有关，这意味着如果你联系技术支持人员并且告诉他们你有一个 X2-2 半机柜配置，支持人员立刻就会知道他们需要知道的有关你的硬件的一切信息，无论对于技术支持人员还是客户来说，这都有益于快速解决问题。

尽管存在这些相似之处，尽管两者有很多共通的特性，Oracle 并不把 Exadata 认为是一个数据仓库设备。一般来说，这是因为 Exadata 提供了这些年来在 Oracle 数据库中已经包含了的所有特性，这是一个包含了全部功能的 Oracle 数据库平台，Exadata 中可以运行当前任何已经运行在 Oracle 数据库中的应用程序，特别是可以应对需要很高并发的混合型负载，而这是一般的数据仓库设备所无法处理的。

■ Kevin 说：当人们描述 Exadata 的时候，到底 Exadata 是不是一个设备是很普遍的困惑。我可以明确地说，Oracle Exadata Database Machine 不是一个设备，但是 Exadata 的存储层确实是由可以称为设备的 Exadata 存储服务器节点组成的。

1.3.2 联机事务处理机器

这种描述是 Exadata 面向更广泛的细分市场的一种宣传策略，虽说也不是特别离谱，但是确实不像其他对于 Exadata 的称呼那样准确，这让我想起一句经典言论：

这要看到底“是”这个词该如何定义

——比尔·克林顿

同样，OLTP（联机事务处理）也是一个没有准确定义的词汇，我们通常使用这个词去描述那种要求低延迟的业务，并且特点是通过索引来进行单块访问。但是也有有一种 OLTP 系统的子集，同样是写密集型，对于支持大量用户并发的需求也非常高。Exadata 并不是为了让写密集型业务尽可能快而设计的解决方案，但值得注意的是，很少有系统可以纯粹地归类于这些类型，大多数系统都是混合型的系统，既包含了长时间运行、对于吞吐量敏感的 SQL 语句，同时也包含了短时间的、对延迟敏感的 SQL 语句。由此引出了看待 Exadata 的另一个视角。

1.3.3 合并平台

这种描述指出了 Exadata 有潜力作为合并多个数据库的平台，从整体成本（TCO）角度看这是可取的，因为它可以降低复杂性（也就是减少了与复杂性相关的成本），又因为减少了需要管理的系统数量，因此降低了管理开销，减少了服务器数量因此又减少了能源消耗和数据中心的成本，并且减少了软件成本和维护费用。这样看待 Exadata 是正确的。由于 Exadata 的特性，它能够充分支持在同一时间的多样化工作负荷，虽然它并不是完美的 OLTP 机器，但是 Flash Cache 功能也仍然为 OLTP 类型的负载提供了确保低延迟的有效机制，同时智能扫描优化方法为要求高吞吐量的 DW 类型工作也提供了卓越的性能，内置的资源管理功能则提供了让这些有点矛盾的需求可以共存于同一平台的能力。事实上，为了不让长时间运行的查询对延迟敏感的负载产生负面影响，很多客户会花很多精力把数据从 OLTP 系统移动到 DW 系统，对混合负载的支持可以完全避免这部分工作，这是混合负载支持能力带来的极大好处之一。在很多案例中，仅仅是将数据从一个平台迁移到另一个平台就会比其他任何操作都要消耗更多的资源，在这方面，Exadata 的能力可以让这一过程在很多情况下都不再必要。

1.4 可选配置

因为 Exadata 是以预先配置整合的系统形式交付的，因此仅仅只有几种可选项。当本书编写时，有 4 种可用版本，被分成两大类，冠以不同的型号名称（X2-2 和 X2-8），这两种型号的存储节点和网络组件是完全相同的，但是在数据库节点配置上则有所不同。

1.4.1 Exadata Database Machine X2-2

X2-2 有 3 种配置：四分之一机柜、半机柜、全机柜。系统是可升级的，比如可以在以后从四分之一机柜升级到半机柜。几种配置的不同之处如下。

四分之一机柜：X2-2 四分之一机柜包含两个数据库服务器和 3 个存储服务器，高容量（high-capacity）版本如果配置成普通冗余（normal redundancy）的话，可以提供大概 33TB 的空闲空间，而高性能（high-performance）版本如果同样配置成普通冗余方式的话，则

提供大约前者的 1/3 或者说大约 10TB 的可用空间。

半机柜：X2-2 半机柜包含 4 个数据服务器和 7 个存储服务器。高容量版本如果配置成普通冗余的话，可以提供大概 77TB 的空闲空间，而高性能版本如果同样配置成普通冗余方式的话，则提供大约 23TB 的可用空间。

全机柜：X2-2 全机柜包含 8 个数据服务器和 14 个存储服务器。高容量版本如果配置成普通冗余的话，可以提供大概 154TB 的空闲空间，而高性能版本如果同样配置成普通冗余方式的话，则提供大约 47TB 的可用空间。

Note：描述一下我们是怎么预估大致的可用空间的。我们取得磁盘的实际容量，减去给 OS/DBFS 使用的 29GB 空间。假设实际磁盘容量在 HC (High-capacity) 模式和 HP (High-Performance) 模式中分别是 1861GB 和 571GB，减去 29G 就分别是 1833GB 和 543GB，然后与机柜中的磁盘数相乘 (36、84 或者 168)，之后再根据冗余度是普通还是高，来除以 2 或者 3，这样就得到了可用空间。注意，asmcmd 报告的“usable free mb”还考虑了如果 failgroup 丢失需要的重新均衡的空间 (req_mir_free_MB)。asmcmd 中的 lsdg 命令是按照如下方法计算的：

$$\text{Free_MB} / \text{redundancy} - (\text{req_mir_free_MB} / 2)$$

半机柜和全机柜配置可以连接到额外的机柜，也就是多机柜配置模式。这样的配置需要额外的 Infiniband 交换机，我们称之为“spine switch”，用来连接额外的机柜，最多可以连接 8 个机柜，根据打算连接的机柜数量可能需要额外的布线。多个机柜的数据库服务器可以整合到一个跨越多机柜的 RAC 数据库中，也可以用来形成几个较小的 RAC 集群。第 15 章将包含连接多个机柜的更多信息。

1.4.2 Exadata Database Machine X2-8

当前只有一个版本的 X2-8，拥有两个数据库服务器节点和 14 个存储节点，可以认为是一个更高效的 X2-2 全机柜，只不过将 X2-2 中的 8 个小型的数据库服务器替换成了两个大型的服务器。如前所述，存储服务器和网络组件是与 X2-2 完全相同的。对于 X2-8 而言，没有什么所谓的升级，如果需要更强的性能，就必须增加另外一台 X2-8，当然，增加额外的存储节点还是有可能的。

1. 升级

四分之一机柜和半机柜可以通过升级来获得更强的能力，报价中包含两种可选的升级方式，半机柜升级到全机柜，或者四分之一机柜升级到半机柜。之所以提供有限的升级选项，是为了保持数据库服务器和存储服务器之间的相对平衡。这些升级都会在现场完成，如果客户下单了升级需求，那么各个单独的组件将会打成一个大包送到客户现场，并且一个 Sun 工程师会安排时间将这些组

件安装到客户现有的机柜中，所有必要的组件都会毫无遗漏地送到现场，包括机柜导轨和线缆。但是令人遗憾的是，线缆的标签似乎并不能及时送到，当我们在自己的实验室里进行升级的时候，由于缺少标签不得不拖延了好几天。

四分之一到半机柜的升级包含了 2 台数据库服务器和 4 台存储服务器，还包括一些额外的 InfiniBand 交换机，这是用来配置成 spine switch 的。半机柜到全机柜的升级包含了 4 台数据库服务器和 7 台存储服务器，但没有额外的 InfiniBand 交换机，因为半机柜就已经包含了所需的 spine switch。

还有一种选项，就是向已有的机柜中添加独立的存储服务器，虽然这不太符合均衡配置的理念，不过 Oracle 还是允许这样做的。不过奇怪的是，他们不支持将存储服务器放在现有的机柜中，即使机柜中还有富裕的空间（比如说在四分之一机柜和半机柜的配置情况下）。

关于升级还有一些值得注意的地方。很多公司都订购了 Exadata V2 系统，现在都正在准备升级，关于升级流程有一些需要解答的问题。不可避免要提到的，是否允许将新的 X2-2 服务器和老的 V2 组件混合在一起呢？答案是肯定的，没问题，可以混合。比如说，在我们的实验环境中，我们就混合 V2（我们最开始的四分之一机柜）和 X2-2 服务器（后来升级为半机柜）。我们选择了将我们现有的四分之一机柜升级到半机柜，而不是另行购买一套独立的四分之一机柜的 X2-2（这也是一种可行的选项）。

另一个经常提到的问题是，对于一个公司，如果他们的存储空间不足了，但是在数据库服务器上还有很充足的 CPU 能力，那么增加额外的存储服务器是不是可行的选项？这个问题并不太容易回答，从许可证角度看，Oracle 是会卖给你额外的存储服务器的，但是请记住，Exadata 的一大目标就是创建一个更加均衡的架构，所以你应该慎重考虑为了应对额外的存储服务器带来的额外吞吐量，你是否需要在数据库服务器上拥有更多的处理能力。不过，如果仅仅是为了处理磁盘空间短缺的问题，购买额外的存储服务器当然是可行的方案。

1.5 硬件组件

读者可能曾经看到过很多与图 1-2 相似的图片，这是 Exadata 全机柜的展示。我们加了一些图示指出在这个盒子里面各部位都是什么组件，本节我们来介绍这些组件。

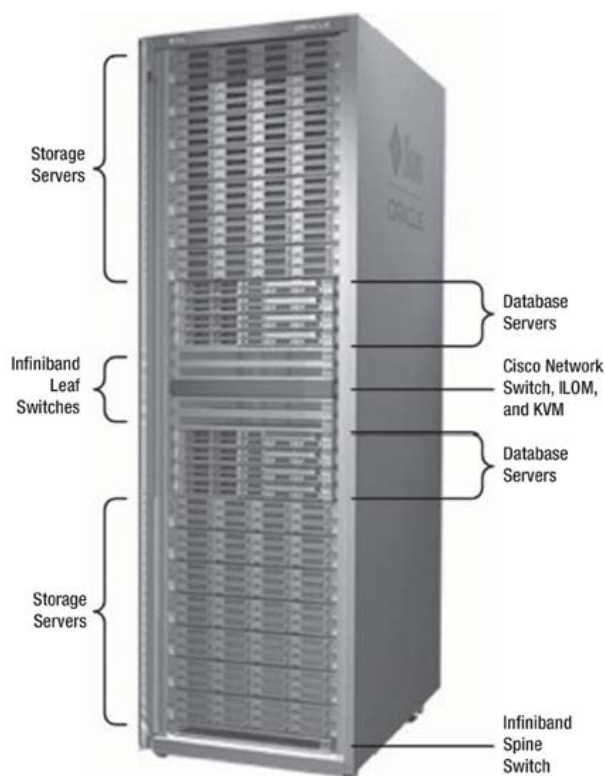


图 1-2 Exadata 全机柜

如图 1-2 所示，大多数网络组件，包括以太网交换机和两个冗余的 InfiniBand 交换机，都放置在机柜的中间部分，这很合理，因为这让连线更简单。在中间部分还有一个 Sun Integrated Lights Out Manager (ILOM) 模块以及 KVM 模块。周围的 8 个插槽是为数据库服务器保留的，剩下的绝大部分都是给存储服务器使用的，除了底部的那一个插槽，是给额外的 InfiniBand “spine” 交换机使用的，用来连接其他的 Exadata 机柜。之所以布置在机柜底部，是因为 Exadata 通常放在数据中心的活动地板上，允许从机柜底部开始布线。

1.5.1 操作系统

目前版本的 X2 硬件配置使用的是基于 Intel 芯片的 Sun 服务器，写作本章时，所有的服务器上预装的都是 Oracle Linux 5，Oracle 宣布将会支持两种版本的 Linux 内核——标准的与 Redhat 兼容的版本和被称为 Unbreakable Enterprise Kernel (UEK) 的加强版内核。这种优化版本，拥有特别适用于 Exadata 的几项改进，其中包括与网络相关的 InfiniBand 使用 RDS 协议的改进。发布 UEK 的一个原因就是 Oracle 可以避开将某些改进融合到标准的开源发布版本中所需要的漫长过程，这样就可以加快 Oracle 对于 Linux 的变化推动。Oracle 一直以来都是 Linux 研发的重要合作伙伴，并且对于代码库做出了不少重大贡献，最终方向也是将 UEK 中包含的所有增强功能都列入标准发行版中。

■ 深入理解 Oracle Exadata

Oracle 也宣布了 X2 数据库服务器将来可以运行 Solaris 11 Express，谈到 Solaris，我们经常被问到 Oracle 是不是有计划推出使用 SPARC CPUs 的 Exadata 版本，至少到本章撰写时，还没有迹象表明 Oracle 会做此决定。看上去 Oracle 似乎将更加持续地推动基于 X86 的解决方案。

X2-2 和 X2-8 两个型号的存储服务器都将只运行在 Oracle Linux 上，Oracle 认为这些服务器是封闭的系统，因此不允许在其上安装任何其他软件。

1.5.2 数据库服务器

当前版本的 X2-2 数据库服务器是基于 Sun Fire X4170 M2 服务器的，每台服务器都有两个 6 核 Intel Xeon X5670 芯片（2.93 GHz）和 96GB 内存，还包含 4 块内置的 300GB 10K RPM SAS 磁盘，另外，网络连接中除了两个 QDR InfiniBand（40Gb/s）端口之外，还包括两个 10Gb 和 4 个 1Gb 的以太网口。注意 10Gb 端口是开启的，你需要提供正确的连接器将其连接入已有的以太网或者光纤网络中。服务器还有一个专门的 ILOM 端口和一对热插拔电源。

X2-8 数据库节点基于 Sun Fire X4800 服务器，这是设计用来处理需要大量内存的系统的，服务器配备了 8 个 8 核 Intel Xeon X7560 处理器（2.26GHz）和 1TB 内存，这样一个全机柜配置总共拥有 128 核和 2TB 内存。

1.5.3 存储服务器

当前版本的 X2-2 和 X2-8 使用的都是相同的存储服务器，每个存储服务器包含了一个 Sun Fire X4270 M2 机器和 12 块磁盘，根据高容量版本或者高性能版本的不同，磁盘可能是 2TB 或者是 600GB 的 SAS 盘。每个存储服务器都拥有 24GB 内存和两个 6 核 Intel Xeon X5670 处理器（2.93GHz），跟 X2-2 数据库服务器中使用的 CPU 是一样的。因为这些 CPU 都属于 Westmere 系列，都内置支持 AES 解压功能，这实质上就提供了硬件支持的加密和解压。每个存储服务器还包含了 4 块 96GB 的 Sun Flash Accelerator F20 PCIe 卡，这就给每个存储节点都提供了总共高达 384GB 的闪存存储。存储服务器上预安装的操作系统是 Oracle Linux 5。

1.5.4 InfiniBand

Exadata 中最重要的硬件组件之一是 InfiniBand 网络，用来在数据库节点和存储节点之间传输数据，如果配置为 RAC 集群，那么还用来作为数据库节点之间的互联（Interconnect）网络使用。除此之外，InfiniBand 网络可以用来连接外部系统，如备份系统。为此，Exadata 提供了两台做冗余的 36 口 QDR InfiniBand 交换机。交换机可以提供每秒 40Gb 的吞吐量。有时候，这些交换机也被称为“Leaf”交换机。另外，每个数据库服务器和存储服务器都配备了双口 QDR InfiniBand 主机通道适配器，除了最小的（四分之一配）的 Exadata 配置之外还都包含第三台 InfiniBand 交换机，这是为了将多个 Exadata 机柜连在一起，这台交换机通常被称为“spine”

交换机。

1.5.5 闪存

正如前面提到的，每个存储服务器上配备了 384GB 的闪存存储，这些存储通常是配置用做缓存的，Oracle 称之为 Exadata 智能闪存（ESFC）。ESFC 的主要目的是最小化单块读的服务时间，此功能提供了大量的磁盘缓存，在一个半机柜配置中大约有 2.5TB。

1.5.6 磁盘

Oracle 为磁盘系统提供了两种选项，一个 Exadata 机器可以选择配置两种模式的磁盘，高容量（high-capacity）或者高性能（high-performance）。如之前提到的，高容量选项提供单盘 2TB、7200RPM 的磁盘，而高性能选项则提供单盘 600GB、15000RPM 的 SAS 磁盘。Oracle 不允许混合使用这两种磁盘类型。由于存储节点上有大量的闪存，因此看上去对于多数大量读的系统，高容量选项就足够了。在我们观察到的混合负载的系统中，闪存在缩减单块读的延迟时间上起到了很大作用。

1.5.7 其他杂项

整套价格中包含了 42U 的机柜，包含了冗余的电源，还包含了以太网交换机。规格表中没有提到以太网交换机的型号，但是截止文本编写时间为止，提供的是 Cisco 的交换机。目前为止，这是整套设备中唯一 Oracle 允许客户自行替换的设备，如果你有另外一个更喜欢的交换机，你可以将已包含的这个交换机替换掉（当然费用自付）。X2-2 还包含了一套 KVM 设备，另外还有一个备用工具包，其中包含了一个额外的闪存卡、一块额外的磁盘和一些额外的 InfiniBand 线缆（在全机柜配置中则是包含两个闪存卡和两块磁盘）。整套价格中不包含给 10Gb 以太网端口使用的 SFP+ 连接器或者线缆，因为这些都没有标准的配置，根据客户网络环境的不同会有所不同，这些端口的用处是将数据库服务器连接入客户的网络中。

1.6 软件组件

组成 Exadata 的软件组件分别位于数据库节点和存储节点中，在数据库节点上运行的是标准的 Oracle 数据库软件，而在存储节点上运行的是 Oracle 新的磁盘管理软件。在这两层中的组件通过 iDB 协议来互相通信，下面的两小节会简要介绍这些软件。

1.6.1 数据库服务器软件

如前面所述,数据库服务器运行的是 Oracle Linux,当然还有另一个选项是运行 Solaris Express,不过到目前为止,我们还没有看到有谁在运行 Solaris。

数据库服务器上还运行着 Oracle 11gR2 软件,这套软件跟运行在其他任何平台上的软件在代码级别并无差异,不是什么特殊的版本,实际上这正是 Exadata 与其他数据仓库一体设备相比卓然不群的功能,这意味着任何可以运行在 Oracle 11gR2 上的应用程序都可以不经任何修改就运行在 Exadata 中。对于那些仅对 Exadata 有效的代码,比如说 iDB, Oracle 也选择将其作为标准发行版的一部分,软件会知道是不是在访问 Exadata 存储,这种感知能力可以让数据库在访问 Exadata 存储时使用到为 Exadata 设计的特有优化手段。

ASM (Oracle 自动存储管理) 是数据库服务器上的整个软件层次中的重要一环,它为 Exadata 存储提供了文件系统和卷管理功能。这是必需的,因为存储设备对于数据库服务器来说并不可见,所以并没有直接的机制可以让数据库服务器去打开和读取 Exadata 存储节点上的文件。ASM 还通过镜像数据块对存储提供了冗余性,可以使用普通冗余(两份 copy)或者使用高度冗余(三份 copy),这是很重要的功能,因为磁盘是物理分布在多个存储服务器中的,ASM 冗余性允许在多个存储节点间镜像数据,因此即使某个存储服务器完全损坏,也不会影响数据库的正常运行。在 Exadata 存储服务器上并没有任何形式的硬件或者软件实现的 RAID 来保护数据,镜像保护功能完全是通过 ASM 来提供的。

虽然通常安装在 Exadata 数据库服务器上的都是 RAC,但这并不是必需的。不过 RAC 确实可以对高可用性和扩展性提供很多好处,对于那些需要比一台服务器能够提供的 CPU 或者内存资源更多的系统而言,RAC 是获取这些额外资源的途径。

数据库服务器和存储服务器通过智能的数据库协议(iDB)来通信,iDB 实现了 Oracle 提出的功能运输(function shipping)架构,这个词汇描述的过程是:iDB 将正在执行的 SQL 语句信息传输到存储节点上,然后将存储节点上计算过的数据(比如说,预过滤的)而并非是数据块直接返回给请求的进程。在这种模式下,iDB 可以将返回给数据库服务器的数据仅仅限制在那些满足查询的行和列中,这种功能传输模式仅仅在执行全表扫描的时候才可用。iDB 也可以用来在 Offloading 不可用或者不适用的时候传输和接收完整的数据块,在这种模式下,iDB 就跟普通的 I/O 协议一样使用,将完整的 Oracle 数据块读出并返回到 Oracle 数据库的 Buffer Cache 中。为了描述地更完整,我们必须提一下,这不是简单的非此即彼的场景,还有很多场景下这两种行为是混合进行的。我们会在第 2 章中进行更详细的讨论。

iDB 在数据库节点和存储节点之间使用可靠数据报套接字(RDS)协议(当然还有 InfiniBand),RDS 是一种低延迟、低开销的协议,与类似于 UDP 这样的协议相比,对 CPU 占用有明显减少,RDS 在 Exadata 出现之前好几年就已经存在了,该协议为进程间通信实现了直接内存访问模型,这就比常规的 TCP 通信降低了延迟,减少了 CPU 开销。

■ Kevin 说：RDS 确实已经存在了不短的时间，RDS 的历史可以回溯到 SilverStorm(后来被 Qlogic 收购) 和 Oracle 之间的合作，当时是为了 Oracle RAC 的 DLM 锁通信，或者更小范围的，为了并行查询数据传输，需要在节点间互联网卡上 (通过 libskgxp) 实现低延迟和高带宽的需求。之后这种模型通过在 PANTASystems 平台 (后来解散了) 中 Oracle 10g 数据库上的 1TB TPC-H 测试值获得验证，再后来 Oracle 又跟 Mellanox 在此协议上紧密合作。

历史故事告诉我们一个重要之处，iDB 是基于 libskgxp 的，这个库从 Oracle 8i 时的早期缓存融合功能开始经过多年的淬炼，在将 SAGE (译者注：Exadata) 投入市场时，这些尝试和技术真正派上了用场。

数据库服务器上的操作系统无法直接看到存储设备，理解这一点很重要，正因为如此，才没有那些用来打开文件、读取数据块，或者执行其他常规功能的系统调用，这也意味着使用标准操作系统程序，比如 iostat，对数据库服务器进行监控是没有帮助的，因为运行着的进程并不会对数据文件进行 I/O 调用。下面展示一些输出结果以印证该论断：

```
KSO@SANDBOX1> @whoami
```

USERNAME	SID	SERIAL#	PREV_HASH_VALUE	SCHEMANAME	OS_PID
KSO	689	771	2334772408	KSO	23922

```
KSO@SANDBOX1> select /* avgskew3.sql */ avg(pk_col) from kso.skew3 a where col1 > 0;
```

...

```
> strace -cp 23922
Process 23922 attached - interrupt to quit
Process 23922 detached
```

% time	seconds	usecs/call	calls	errors	syscall
49.75	0.004690	0	10902	5451	setsockopt
29.27	0.002759	0	6365		poll
11.30	0.001065	0	5487		sendmsg
9.60	0.000905	0	15328	4297	recvmsg
0.08	0.000008	1	16		fcntl
0.00	0.000000	0	59		read
0.00	0.000000	0	3		write
0.00	0.000000	0	32		open

■ 深入理解 Oracle Exadata

0.00	0.000000	0	20	close
0.00	0.000000	0	4	stat
0.00	0.000000	0	4	fstat
0.00	0.000000	0	52	lseek
0.00	0.000000	0	33	mmap
0.00	0.000000	0	7	munmap
0.00	0.000000	0	1	semctl
0.00	0.000000	0	65	getrusage
0.00	0.000000	0	32	times
0.00	0.000000	0	1	semtimedop

100.00	0.009427		38411	9760 total

在上面的输出中，我们对一个用户后台进程（有时候也称为影子进程）运行了 `strace`，这是负责给用户获取数据的进程，正如你们看见的，`strace` 抓取到的最大量的系统调用是跟网络相关的（`setsockopt`、`poll`、`sendmsg` 和 `recvmsg`）。相比之下，非 Exadata 平台上，我们大多看到磁盘 I/O 相关的事件，主要是一些 `read` 调用的形式。下面是一些非 Exadata 平台上的例子：

```

KSO@LAB112> @whoami

USERNAME          SID      SERIAL#  PREV_HASH_VALUE  SCHEMANAME  OS_PID
-----
KSO                249      32347    4128301241  KSO         22493

KSO@LAB112> @avgskew

AVG(PK_COL)
-----
16093749.8

...

[root@homer ~]# strace -cp 22493
Process 22493 attached - interrupt to quit
Process 22493 detached
% time      seconds  usecs/call   calls   errors syscall
-----
88.86      4.909365    3860      1272         0  pread64
10.84      0.599031      65      9171         0  gettimeofday
0.16      0.008766      64       136         0  getrusage
0.04      0.002064      56        37         0  times
0.02      0.001378    459         3         0  write

```

0.02	0.001194	597	2	statfs
0.02	0.001150	575	2	fstatfs
0.02	0.001051	350	3	read
0.01	0.000385	96	4	mmap2
0.00	0.000210	105	2	io_destroy
0.00	0.000154	77	2	io_setup
0.00	0.000080	40	2	open
0.00	0.000021	11	2	fcntl64

100.00	5.524849	10638		total

可以看到在非 Exadata 平台上抓取的主要系统调用是 I/O 相关的（pread64），上面两段代码的主旨是展示在 Exadata 中，访问磁盘中的数据所使用的方法与常规情况很不相同。

1.6.2 存储服务器软件

Cell Services（cellsrv）是运行在存储节点上的主要软件，这是一个多线程的程序，用于处理从数据库服务器上发送过来的 I/O 请求，根据不同的请求，cellserv 会返回处理过的数据或者完整的数据块。cellsrv 还会实现资源管理器定义的 I/O 分布规则，保证 I/O 正确分配给不同的数据库和用户组。

在 Exadata 存储节点上还会运行两个其他的程序。其中之一的管理服务器（MS）是一个 Java 程序，提供 cellsrv 和 Cell 命令行（cellcli）程序之间的接口，MS 还同时提供 cellsrv 和 Grid Control Exadata 插件（以一系列通过 rsh 来运行的 cellcli 命令来实现）之间的接口。第二个程序是重启服务器（RS），RS 实际上是一系列进程，负责监控其他进程并在必要的时候重启这些进程。OSWatcher 也安装在存储节点上，用标准的 UNIX 程序（比如 vmstat 和 netstat）收集操作系统的统计信息历史。要注意，Oracle 不授权在存储服务器上安装任何其他额外的软件。

当你第一次遇见 Exadata 的时候，你最想做的第一件事情大概就是登录到存储节点上，看看到底运行着什么，不过很遗憾，存储服务器通常不是每个账号都可以登录的，只有系统管理员或者 DBA 们才可以登录。下面列出了在一个运行着的存储服务器上用 ps 命令返回的结果：

```
> ps -eo ruser,pid,ppid,cmd
```

```
RUSER      PID  PPID  CMD
root       12447    1 /opt/oracle/.../cellsrv/bin/cellrssrm -ms 1 -cellsrv 1
root       12453 12447 /opt/oracle/.../cellsrv/bin/cellrsbmt -ms 1 -cellsrv 1
root       12454 12447 /opt/oracle/.../cellsrv/bin/cellrsmmt -ms 1 -cellsrv 1
root       12455 12447 /opt/oracle/.../cellsrv/bin/cellrsomt -ms 1 -cellsrv 1
root       12456 12453 /opt/oracle/.../bin/cellrsbkm
           -rs_conf /opt/oracle/.../cellsrv/deploy/config/cellinit.ora
```

■ 深入理解 Oracle Exadata

```

                                -ms_conf /opt/oracle/cell
root      12457 12454 /usr/java/jdk1.5.0_15//bin/java -Xms256m -Xmx512m
                                -Djava.library.path=/opt/oracle/.../cellsrv/lib
                                -Ddisable.checkForUpdate=true -jar /opt/oracle/cell11.2
root      12460 12456 /opt/oracle/.../cellsrv/bin/cellrsm
                                -rs_conf /opt/oracle/.../cellsrv/deploy/config/cellinit.ora
                                -ms_conf /opt/oracle/cell
root      12461 12455 /opt/oracle/.../cellsrv/bin/cellsrv 100 5000 9 5042
root      12772 22479 /usr/bin/mpstat 5 720
root      12773 22479 bzip2 --stdout
root      17553      1 /bin/ksh ./OSwatcher.sh 15 168 bzip2
root      20135 22478 /usr/bin/top -b -c -d 5 -n 720
root      20136 22478 bzip2 --stdout
root      22445 17553 /bin/ksh ./OSwatcherFM.sh 168
root      22463 17553 /bin/ksh ./oswsub.sh HighFreq ./Exadata_vmstat.sh
root      22464 17553 /bin/ksh ./oswsub.sh HighFreq ./Exadata_mpstat.sh
root      22465 17553 /bin/ksh ./oswsub.sh HighFreq ./Exadata_netstat.sh
root      22466 17553 /bin/ksh ./oswsub.sh HighFreq ./Exadata_iostat.sh
root      22467 17553 /bin/ksh ./oswsub.sh HighFreq ./Exadata_top.sh
root      22471 17553 /bin/bash /opt/oracle.cell/ExadataDiagCollector.sh
root      22472 17553 /bin/ksh ./oswsub.sh HighFreq
                                /opt/oracle.oswatcher/osw/ExadataRdsInfo.sh
root      22476 22463 /bin/bash ./Exadata_vmstat.sh HighFreq
root      22477 22466 /bin/bash ./Exadata_iostat.sh HighFreq
root      22478 22467 /bin/bash ./Exadata_top.sh HighFreq
root      22479 22464 /bin/bash ./Exadata_mpstat.sh HighFreq
root      22480 22465 /bin/bash ./Exadata_netstat.sh HighFreq
root      22496 22472 /bin/bash /opt/oracle.oswatcher/osw/ExadataRdsInfo.sh HighFreq

```

可以看到，有很多进程都类似于 `cellsrv × × ×`，这些是组成重启服务器的进程。请注意第一行黑体标注的进程，这就是我们提到的管理服务器的 Java 程序，第二行黑体标注的进程是 `cellsrv` 本身，最后，你可以看到跟 `OSwatcher` 相关的一系列进程。你应该还可以看到所有的进程都是以 `root` 用户启动的，虽然在存储服务器上还有其他一些半特权用户，不过很显然，这个系统并不是需要太多用户登录的。

另外一个查看相关进程的有趣方法是使用 `ps -H` 命令，可以缩进列表的样式显示这些进程互相是如何关联的。当然，通过进程 ID (PID) 和父进程 ID (PPID) 之间的关系，我们手动也可以做出这样的树形结构，不过 `-H` 选项让这项工作容易多了。下面是编辑过的 `ps -H` 命令的输出结果：

```

cellrsm =<= main Restart Server
    cellrsm
        cellrsm

```

```
cellrsmmt
cellrsmmt
  java - .../oc4j/ms/j2ee/home/oc4j.jar <= Management Server
cellrsomt
  cellsrv
```

查看存储服务器上哪些资源正在被消耗也同样很有意思，下面是 top 命令的输出：

```
top - 18:20:27 up 2 days, 2:09, 1 user, load average: 0.07, 0.15, 0.16
Tasks: 298 total, 1 running, 297 sleeping, 0 stopped, 0 zombie
Cpu(s): 6.1%us, 0.6%sy, 0.0%ni, 93.30%id, 0.3%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 24531712k total, 14250280k used, 10281432k free, 188720k buffers
Swap: 2096376k total, 0k used, 2096376k free, 497792k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
12461	root	18	0	17.0g	4.5g	11m	S	105.9	19.2	55:20.45	cellsrv
1	root	18	0	10348	748	620	S	0.0	0.0	0:02.79	init
2	root	RT	-5	0	0	0	S	0.0	0.0	0:00.14	migration/0
3	root	34	19	0	0	0	S	0.0	0.0	0:01.45	ksoftirqd/0
4	root	RT	-5	0	0	0	S	0.0	0.0	0:00.00	watchdog/0

top 的输出显示了 cellsrv 的使用率超过了一块 CPU 核（超过了 100%），这在繁忙的系统中很常见，因为 cellsrv 进程本来就是多线程的。

1.7 软件架构

在本节中，我们将简要讨论一些主要的软件组件以及它们是如何在 Exadata 架构中互相连接的，有一些组件是同时运行在数据库服务器和存储服务器上的。

图 1-3 描述了 Exadata 平台的整体架构。

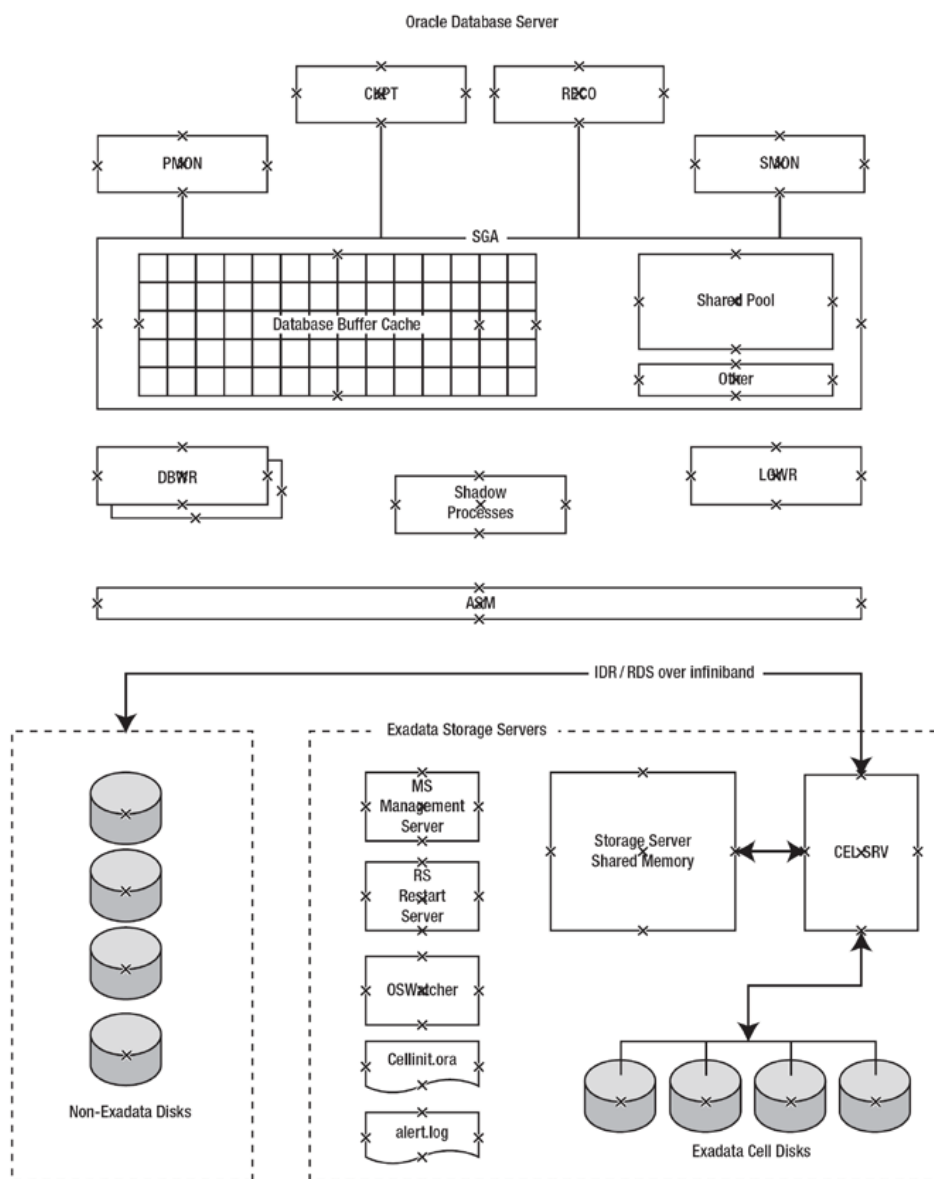


图 1-3 Exadata 架构图

图 1-3 中上半部分显示了一台数据库服务器中的主要组件，下半部分则显示了一台存储服务器中的主要组件。上半部分看上去应该非常熟悉，这就是一个标准的 Oracle 11g 架构，显示了包含缓冲区和共享池的系统全局区（SGA），也显示了一些关键进程，比如日志写进程（LGWR）和数据库写进程（DBWR），当然还有其他很多进程。在共享内存中也可以细化出更多详细的布局，不过这应该已经足以展示一个数据库服务器上的基本框架了。

下半部分显示了一台存储服务器的组件，存储服务器的架构很简单，实际上只有一个进程（cellsrv）处理和数据库服务器之间的所有通信，还有极少数的辅助进程管理和监控环境。

在架构图中有一点你可能注意到了，`cellsrv` 使用了 `init.ora` 并且还有 (`alert log`)。实际上，存储软件跟 Oracle 数据库有惊人的相似之处，当然这也在情理之中。`cellinit.ora` 文件中包含了一系列在 `cellsrv` 启动时将会设置的参数。告警日志 (`alert log`) 则是用来记录需要注意的事件的，这跟 Oracle 数据库中的告警日志很相似。另外，自动诊断信息库 (ADR) 也包含在存储软件中，用来捕捉和报告诊断信息。

读者还可以从图 1-3 中看到有一个独立的进程没有连接到任何数据库实例 (DISKMON)，这个进程是为了执行与 Exadata 存储相关的一些任务，虽然被称为 DISKMON，但实际上却是网络和节点监控的进程，作用是确认这些节点的存活状态。DISKMON 还负责将数据库资源管理器 (DBRM) 的计划传送给存储服务器，另外，DISKMON 对于每个实例都有一个子进程，用来负责 ASM 和数据库之间的通信。

数据库服务器和存储服务器之间的连接通过 InfiniBand 卡，两层之间的所有通信都通过这种传输机制来完成，这包括了通过 DBWR 进程和 LGWR 进程的写请求和其他用户后台进程的读请求。

图 1-4 从另一个视角展示了整体架构，这个视角关注于整个软件栈以及在数据库网格和存储网格中的软件跨多个服务器的分布方式。

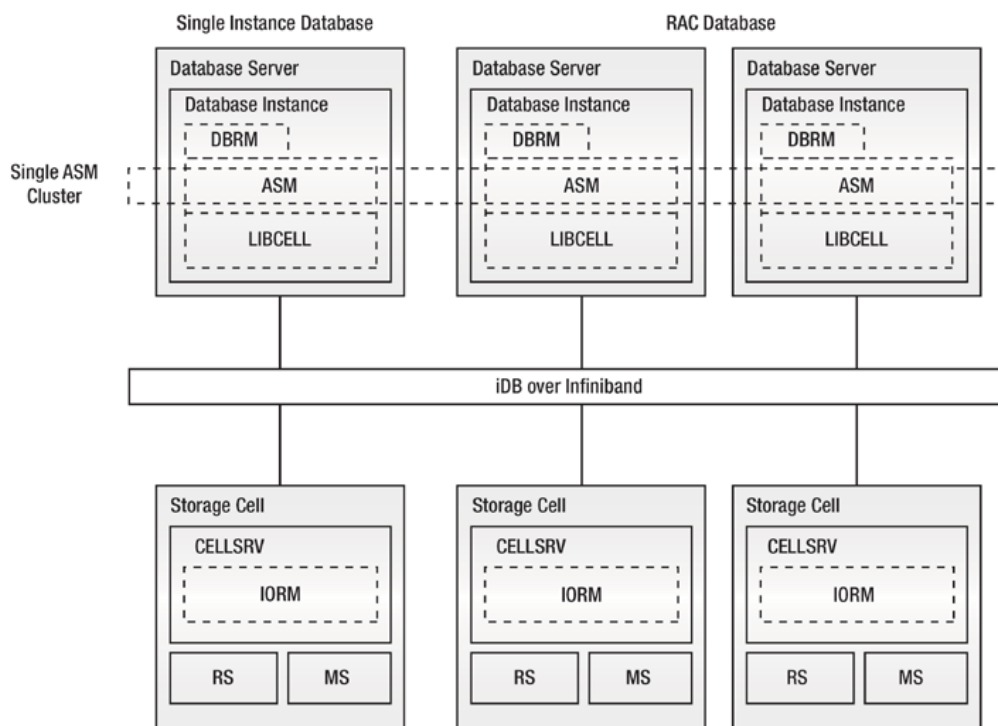


图 1-4 Exadata 另一个视角架构

正如我们探讨过的，ASM 是关键组件，我们将它画成横跨在两层之间所有通信线路上的一个虚线框对象，这是为了指出 ASM 提供了文件和数据库所知的存储层对象之间的映射。ASM 实际上并不是挡在存储和数据库之间，在整个软件栈中也并不是每一次进程进行“磁盘访问”都必须经

过的一层步骤。

图 1-4 还显示了运行在数据库实例中的数据库资源管理器（DBRM）和运行在存储服务器上的 cellsrv 中的 I/O 资源管理器（IORM）的关系。

在图 1-4 中最后重要的组件是 LIBCELL，这是一个跟 Oracle Kernel 相关联的库，LIBCELL 中的代码知道如何通过 iDB 协议来请求数据，这提供了一个完全无干扰的机制，允许 Oracle 内核通过基于网络的调用而不是操作系统的读写来与存储层通信。iDB 是在 OpenFabrics Enterprise Distribution 组织发布的 RDS 协议之上实现的，这是一种低延迟、低 CPU 消耗的协议，提供了进程间通信的机制。你可能还会看到这个协议在某些 Oracle 的市场宣传资料中被称为是零丢失零拷贝（ZDP）InfiniBand 协议。图 1-5 是一个基本原理图示，说明了为什么 RDS 协议比传统的基于 TCP 的协议（比如 UDP），要更高效。

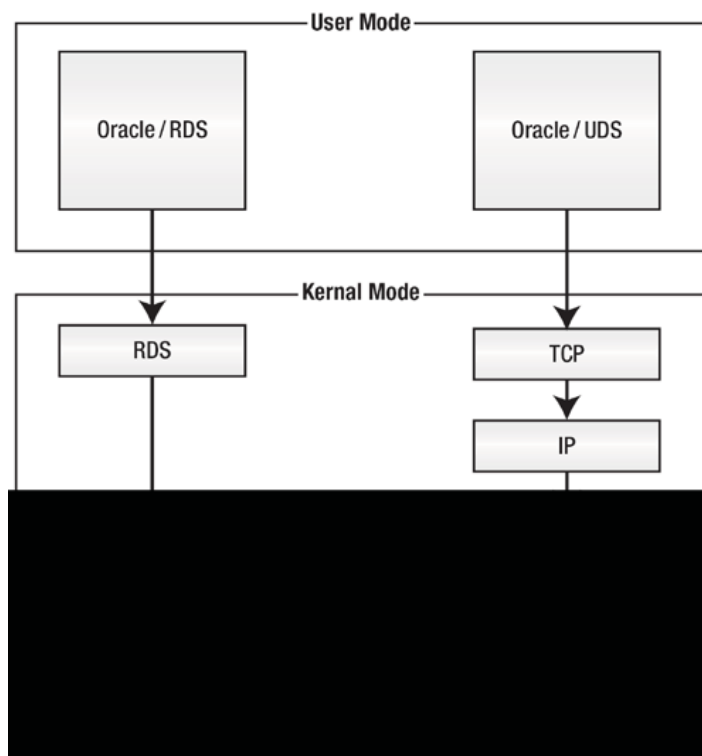


图 1-5 RDSschematic

从图中可以看到，使用 RDS 协议跳过了 TCP 流程中需要在网络上传输数据的那部分开销，需要注意的是 RDS 协议也用于 RAC 节点之间的互联通信。

1.8 总结

Exadata 是硬件和软件紧密结合的产品组合，有关硬件本身其实没什么神奇的地方，大多数的性能优势来自于整合的组件以及在存储层上实现的软件。在下一章，我们将深入介绍卸载（Offloading）概念，这是 Exadata 之所以显著区别于其他运行 Oracle 数据库的平台的关键点。

第 2 章

卸载/智能扫描

卸载（Offloading）是 Exadata 的独门武器，正是它让 Exadata 不同于其他任何一种运行 Oracle 的平台。卸载指的是将处理能力从数据库服务器转移到存储层，它也正是 Exadata 平台提供的主要卖点，它不仅仅转移了 CPU 的使用，更主要的好处是减少了那些必须要返回给数据库服务器的数据量，而这正是大多数大型数据库的主要瓶颈所在。

卸载这个词在某些方面可以跟智能扫描（Smart Scan）互通，我们认为卸载是更好的描述方式，因为它指出了一部分传统上由数据库处理的 SQL 流程现在可以从数据库层面“卸载（offloaded）”到存储层面。这是一个更普适的措辞，它可以被用在那些甚至与 SQL 处理并无关系的优化上，比如用于提高备份和恢复操作的效率。

而另一方面，智能扫描，是一个更狭义的词汇，仅仅是指 Exadata 对 SQL 语句的优化方式，当进行扫描操作（通常是全表扫描）的时候，这些优化就会起作用。对于智能扫描更明确的定义也体现在 Oracle 核心代码中对于智能扫描相关等待事件的命名上，实际上有两个等待事件在名称中包含了“Smart Scan”字样，即“Cell Smart Table Scan”和“Cell Smart Index Scan”。我们将在后续的第 10 章里更详细地探讨这两个等待事件。好吧，我承认“智能扫描”这个词确实带有一些市场推广的成分，但是在代码上也确实有等待事件以此命名。无论如何，虽然这两个词在某些程度上可以互通，但是请记住“卸载”指的可不仅仅是加速 SQL 语句的执行。

本章我们将主要讲述“智能扫描”这种优化方式。我们会提到跟智能扫描相关的多种优化手段，它们是如何工作的，以及如果想使用智能扫描，那么有哪些前提条件必须满足。我们也会讲述一些技术，用以帮助你确认对于给定的 SQL 语句，智能扫描有否被使用到。至于在本书别的章节会提到的其他卸载优化，本章仅仅会做简要描述。

2.1 为何卸载如此重要

我们无法更加强调这个概念是何等重要，将数据库的处理能力转移到存储层这样的理念绝对是一个巨大的飞跃。这种概念已经存在了一段时间，实际上在几年前就有传言说 Oracle 带着这种想法跟至少一家最大的 SAN 制造厂商进行过密谈。显然那时候存储制造商对此想法并无兴趣，于是 Oracle 决定靠一己之力来推进这个理念。随后 Oracle 与惠普（HP）合作推出了最开始的 Exadata V1，集成了卸载的概念。没过几年，Oracle 就出手收购了太阳微系统公司（Sun Microsystems），这使得

■ 深入理解 Oracle Exadata

Oracle 一举成为可以全面提供软硬件整合解决方案的公司，这也让 Oracle 可以完全控制到底想将哪些功能整合到产品里去。

卸载的重要性在于，当前大型数据库中一大主要瓶颈就是在磁盘系统和数据库服务器之间传输必要的能够满足数据仓库类型的查询所消耗的大量时间（因为带宽所限）。这一方面是硬件架构的问题，但是更大的问题在于传统 Oracle 数据库所传输的大量数据。Oracle 数据库处理数据非常快也非常聪明，但是对于那些要读取大量数据的查询，将这些数据取到数据库来仍然会消耗很长时间。所以就跟任何优秀的性能调整专家一样，Oracle 开始致力于减少最消耗时间的那些操作所花的时间。在分析中他们意识到每个需要磁盘读的查询会因为有大量数据要返回并由数据库服务器来处理而显得特别低效。Oracle 研发了最好的缓存管理机制，但是对于那些确实庞大的数据集，将所有数据都保留在数据库服务器的内存中实在是不切实际的。

■ Kevin 说：作者们很好地解读了 Oracle 查询处理的发展历程。不过我还是想提醒各位，当今商用的 x64 服务器已经不再在架构上限制内存配置了。比如基于具有 QuickPath 互联功能的 Intel Xeon 7500 处理器的服务器就支持在多 DIMM 插槽上的大量内存通道，商用服务器具有数 TB 内存已然非常普遍。实际上，X2-8 Exadata 就支持在数据库网格上拥有 2TB 内存，并且此能力随着时间推移必然会不断增强。走向极大内存的 x64 系统的潮流方兴未艾，我期待本书能保留足够长的时间，当以后的读者回过头来再看时会发现此注解所言不假。关于 Exadata 需要记住的重要一点是，它拥有 Oracle 数据库的所有功能并且附加了 Exadata 存储服务器，比如在某些场合，为了满足服务水平，正确的解决方案是尽量完全排除磁盘介质的低性能，客户就可以选择将深度压缩（比如 Exadata 混合列式压缩）和内存间并行查询（In-Memory Parallel Query）一起使用。

想像一下你能想到的最快查询：单张表中一条记录中的一列，并且你还知道这条记录存储在哪里（通过 rowid）。在传统 Oracle 数据库中，为了获取这列的值最少也要有一个数据块被读入内存（通常是 8K 大小），让我们假设该表每个数据块中平均存储着 50 条记录，那么很简单，你传输了额外的 49 行到数据库服务器中，对于此查询来说这就是无谓的开销。如果有一亿次呢？你应该开始意识到在大型数据仓库中这个问题的严重性了。消减在存储和数据库服务器之间传递完全不需要的数据所花费的时间，正是 Exadata 想要解决的主要问题。

卸载就是用来解决这样的问题——在各层之间传输无关数据而花费过多时间。卸载有三个设计目标，不过主要目标的重要性远超其他两个。

- 减少磁盘系统和数据库服务器之间的传输数据量。
- 减少数据库服务器上的 CPU 占用。
- 减少存储层磁盘读取时间。

减少数据量是主要诉求和主要目标。卸载操作带来的大多数优化手段都是为了实现该目标。减

少 CPU 占用同样很重要，但这不是 Exadata 所带来的主要好处，因此只好位居于减少传输数据量之后了（不过你们将会看到，解压是一个例外，因为解压操作会发生在存储服务器上）。另外还有几个降低磁盘读取时间的优化措施，不过，即使其中一些结果也颇引人注目，我们仍然不认为这一项目标是 Exadata 的最根本诉求。

Exadata 是一个软硬件一体的产品，依靠这两方面组件，提供了令人瞩目的超越非 Exadata 平台的性能提升，不过，相比而言，软件方面带来的性能提升让硬件带来的好处相形见绌。下面是一个例子：

```
SYS@SANDBOX> alter session set cell_offload_processing=false;
```

```
Session altered.
```

```
Elapsed: 00:00:00.06
```

```
SYS@SANDBOX> select count(*) from kso.skew3 where col1 < 0;
```

```

COUNT(*)
-----
          2
1 row selected.
```

```
Elapsed: 00:00:51.09
```

```
SYS@SANDBOX> alter session set cell_offload_processing=true;
```

```
Session altered.
```

```
Elapsed: 00:00:00.07
```

```
SYS@SANDBOX> select count(*) from kso.skew3 where col1 < 0;
```

```

COUNT(*)
-----
          2
1 row selected.
```

```
Elapsed: 00:00:00.15
```

该示例演示了在一个具有 3.84 亿行记录的单表上的扫描。我们第一次是在禁用了卸载以后运行的，用到了 Exadata 所有硬件所能提供的好处但是没有软件的，你会发现即使是在 Exadata 硬件上，这条查询也执行了将近 1 分钟。注意这是仅仅分散到我们这台 V2 四分之一机柜的三台存储服务器上并且也没有使用任何闪存。然后我们重新启用了卸载，查询不到 1 秒钟就完成了。很显然在两次执行中硬件都是一样的，那么无疑正是卸载操作这样的软件能力带来了如此大的差别。

一个大众版的 Exadata ?

经常会看到关于搭建一个大众版 Exadata 的主题。该想法是搭建一套在某些方面模仿 Exadata 的硬件平台，想来也是为了能花费比 Oracle 对 Exadata 的收费要低的成本。当然这些建议都是在克隆 Exadata 的硬件部分，因为软件模块无法复制（仅仅是这点就应该让你不用再问到底这种尝试是否切实可行了）。不过，搭建自己的 Exadata 听上去很有吸引力，因为购买单独硬件模块的开销会比 Oracle 提供的整合价格要低，但是这种想法有几个缺点：

1. 硬件模块中获得最多关注的是闪存。你可以购买一个拥有很大缓存的 SAN 或者 NAS。中等尺寸的 Exadata（1/2 机架）在存储服务器上提供大约 2.5TB 的闪存，这确实是一个很大的数字，但是缓存了什么跟缓存有多大同样重要。Exadata 足够智能到不会去缓存那些不大可能从缓存中收益的数据，比如，缓存镜像块就没什么用处，因为 Oracle 只会从主块中读取数据（除非发现主块损坏）。Oracle 在编写管理缓存的软件上已经历时弥久，所以不用惊讶它的卓越表现，当扫描一个大表的时候，Oracle 不会缓存所有的数据块，这样，那些经常会被读取的数据块也就还会留在内存中。如果没有这种了解数据库的缓存算法，普通的 SAN 或者 NAS 必须要配备大得多的缓存才可以与 Exadata 的闪存一较高下。另外请记住，在非 Exadata 存储上保存的数据量也要大很多，因为你无法使用混合列式压缩。

2. 从硬件方面看更重要的，但是很奇怪却又偶尔会被 DIY 建议所忽略的，是存储和数据库之间的数据吞吐量。Exadata 一体机在存储和数据库服务器之间提供了比当今大多数硬件架构都要平衡的数据通道，所以第二个关注的领域通常是各层之间的带宽，但是增加各层之间的数据吞吐量却并非像听上去这么简单。Exadata 通过 InfiniBand 和可靠数据报套接字（Reliable Datagram Sockets，RDS）协议来增大吞吐量，Oracle 研发了运行在 InfiniBand 网络上的 iDB 协议，而 iDB 协议在运行于非 Exadata 硬件的数据库中是不可用的，因此如果想要增加数据层之间的带宽，则必须要采用其他的方法，所以你可以使用运行在万兆网上的 IPOB、iSCSI 或者 NFS，又或者使用高速的光纤连接方案，无论哪种方案在服务器上都需要多块接口卡（都需要连接到快速系统总线）。存储设备还需要提供足够的数据输出量以适应通路的消费能力（这正是当 Oracle 提到平衡配置时所指的意思）。同时，你还需要决定采用什么硬件，还要测试所有环节来确保它们能良好协作，不会在磁盘到数据库服务器之间有任何一处瓶颈产生。

3. 第三点通常会被 DIY 方案提到的是数据库服务器自身。Exadata 的硬件规格随处都可以查到，所以去购买完全相同的 Sun 机器是很容易做到的。但是很不幸，你需要计划购买更多的 CPU，因为你无法将 CPU 的处理能力卸载到 Exadata 存储服务器上，而随之就会带来 Oracle 数据库许可证费用的增加。

4. 假设我们可以做到在硬件的每个部分都和 Exadata 硬件性能相当，我们还是无法期盼能够获得与 Exadata 相近的整体性能，因为软件才是 Exadata 性能提升的点睛之笔，通过在 Exadata 中禁用卸载来做比较测试可以很容易地证明这点，我们会看到没有了软件助力的硬件性能到底怎样。Exadata 软件做的事情很大一部分就是消除完全不必要的工作，比如把最终将被丢弃的列和行仍然传输到数据库服务器中。

就像我们的朋友 Cary Millsap 常喜欢说的那样，“做一件事情最快的方法就是不要去做它”。

2.2 卸载包含了什么

有很多优化手段可以被归类到卸载的大旗之下，本章我们主要讲述通过智能扫描（Smart Scan）实现的 SQL 语句优化。智能扫描的三大优化方法是：字段投影、谓词过滤、存储索引。大多数智能扫描优化方法的主要目标就是在执行扫描时减少需要传输回数据库服务器端的数据量，当然，有些优化也尝试卸载 CPU 密集型操作，比如说解压缩。本章我们不会过多提及与 SQL 语句处理无关的优化，比如智能文件创建（Smart File Creation）以及同 RMAN 相关的优化，这些主题在本书的其他章节中会做更详细的描述。

■ Kevin 说：Offload 处理流程看上去非常复杂，作者们正确地指出了智能扫描带来的最大好处是降低存储和数据库服务器之间的数据传输，但是在存储节点（Storage Cell）中解压 Exadata 混合列式压缩数据也能带来 CPU 卸载的好处，但是经过这样的 Offload 处理，实际上是提高了存储和数据库之间的数据传输量，因此如何权衡很重要，即使实际上在解压以后更多的数据会被传输到数据库服务器上，在存储节点上解压 EHCC 数据（过滤之后的）也仍然是有意义的。所有的技术解决方案实际上都是在博弈。

2.2.1 字段投影

字段投影是指在 Exadata 中通过只返回感兴趣的列（指那些在 SELECT 列表中或者必须参加数据库层 JOIN 操作的列）来限制在存储层和数据库层之间的数据传输。如果查询从拥有 100 个字段

■ 深入理解 Oracle Exadata

的表中请求 5 个字段的值，Exadata 能够消除非 Exadata 存储原本会返回给数据库层的大部分数据。此功能所做的工作远超出你可能的预期，在响应时间上会带来非常巨大的影响，下面是一个例子。

```
SYS@SANDBOX1> alter system flush shared_pool;

System altered.

Elapsed: 00:00:00.12
SYS@SANDBOX1> alter system flush buffer_cache;

System altered.

Elapsed: 00:00:00.13
SYS@SANDBOX1> alter session set "_serial_direct_read"=true;

Session altered.

Elapsed: 00:00:00.00
SYS@SANDBOX1> alter session set cell_offload_processing=false;

Session altered.

Elapsed: 00:00:00.01
SYS@SANDBOX1> select count(col1) from kso.skew3;

COUNT(COL1)
-----
      384000044

1 row selected.

Elapsed: 00:00:51.32
SYS@SANDBOX1> alter session set cell_offload_processing=true;

Session altered.

Elapsed: 00:00:00.00
SYS@SANDBOX1> select count(col1) from kso.skew3;

COUNT(COL1)
-----
```

```
384000044
```

```
1 row selected.
```

```
Elapsed: 00:00:26.27
```

这个例子值得讨论一下。首先我们通过设置 `_SERIAL_DIRECT_READ` 参数强制使用直接路径读取（后续会有更多描述），然后我们设置 `CELL_OFFLOAD_PROCESSING` 为 `FALSE`，禁用了智能扫描。我们的测试语句并没有 `WHERE` 子句，这意味着谓词过滤和存储索引都无法被使用来减少从存储层传输过来的数据，因为这两种优化方式只有在有 `WHERE` 子句时才生效（我们稍后就讨论这两种优化），所以只剩下字段投影这种优化了。当看到仅仅只有字段投影就将响应时间缩短了一半时，你会不会感到惊喜？必须承认，当第一次亲眼目睹的时候，我们被震撼了，但是仔细想一下却又觉得理所当然。你应该知道必须返回给数据库服务器的列并不仅仅是在 `SELECT` 列表中的字段，这是一个非常普遍的误解，实际上在 `WHERE` 条件中 `JOIN` 的列也必须被返回。事实上，在早期版本的 Exadata 中，字段投影功能并不像原先设计的那样有效，它会返回包含在 `WHERE` 条件中的所有字段，在很多情况下都包含了不必要的字段。

可以使用 `DBMS_XPLAN` 包来显示字段投影的信息，虽然在默认情况下并不会显示。投影的数据也同样保存在 `V$SQL_PLAN` 表的 `PROJECTION` 字段中。下面是一个例子。

```
SYS@SANDBOX> select count(s.col1),avg(length(s.col4))
  2  from kso.skew s, kso.skew2 s2
  3  where s.pk_col = s2.pk_col
  4  and s.col1 > 0
  5  and s.col2='asddsadasd';
```

```
COUNT(S.COL1) AVG(LENGTH(S.COL4))
```

```
-----
127999992          1
```

```
1 row selected.
```

```
SYS@SANDBOX> select sql_id, child_number, sql_text
  2  from v$sql where sql_text like '%skew%';
```

```
SQL_ID          CHILD SQL_TEXT
```

```
-----
8xa3wjh48b9ar      0 select count(s.col1),avg(length(s.col4)) from kso.skew s, kso.
```

```
1 row selected.
```

■ 深入理解 Oracle Exadata

```
SYS@SANDBOX> select * from
  2 table(dbms_xplan.display_cursor('&sql_id','&child_no','+projection'));
Enter value for sql_id: 8xa3wjh48b9ar
Enter value for child_no:
```

PLAN_TABLE_OUTPUT

SQL_ID 8xa3wjh48b9ar, child number 0

```
select count(s.col1),avg(length(s.col4)) from kso.skew s, kso.skew2 s2
where s.pk_col = s2.pk_col and s.col1 > 0 and s.col2='asddsadasd'
```

Plan hash value: 3361152066

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
0	SELECT STATEMENT					360K(100)	
1	SORT AGGREGATE		1	30			
* 2	HASH JOIN		64M	1836M	549M	360K (1)	01:12:02
* 3	TABLE ACCESS STORAGE FULL	SKEW	16M	366M		44585 (2)	00:08:56
4	TABLE ACCESS STORAGE FULL	SKEW2	128M	732M		178K (1)	00:35:37

Predicate Information (identified by operation id):

- 2 - access("S"."PK_COL"="S2"."PK_COL")
- 3 - storage(("S"."COL2"='asddsadasd' AND "S"."COL1">0))
 - filter(("S"."COL2"='asddsadasd' AND "S"."COL1">0))

Column Projection Information (identified by operation id):

- 1 - (#keys=0) COUNT(LENGTH("S"."COL4"))[22], COUNT("S"."COL1")[22],
 - SUM(LENGTH("S"."COL4"))[22]
- 2 - (#keys=1) "S"."COL4"[VARCHAR2,1], "S"."COL1"[NUMBER,22]
- 3 - "S"."PK_COL"[NUMBER,22], "S"."COL1"[NUMBER,22], "S"."COL4"[VARCHAR2,1]
- 4 - "S2"."PK_COL"[NUMBER,22]

33 rows selected.

```

SYS@SANDBOX> select projection from v$sql_plan
2 where projection is not null
3 and sql_id = '8xa3wjh48b9ar';

```

PROJECTION

```

-----
(#keys=0) COUNT(LENGTH("S"."COL4"))[22], COUNT("S"."COL1")[22], SUM(LENGTH("S"."COL4"))[22]
(#keys=1) "S"."COL4"[VARCHAR2,1], "S"."COL1"[NUMBER,22]
"S"."PK_COL"[NUMBER,22], "S"."COL1"[NUMBER,22], "S"."COL4"[VARCHAR2,1]
"S2"."PK_COL"[NUMBER,22]

```

4 rows selected.

如上你可以看到，执行计划的输出中显示了投影的信息，但是必须要在调用 DBMS_XPLAN 包的时候使用+PROJECTION 参数。请注意两张表中的 PK_COL 字段都列在了 PROJECTION 部分，但是并不包含 WHERE 条件中的所有字段，只有那些需要返回给数据库端的列（连接的列）才被列入。同样需要注意，投影信息并不是 Exadata 特有的，它是全数据库通用的。

V\$SQL 视图中有字段定义了因为卸载节约的数据量(IO_CELL_OFFLOAD_ELIGIBLE_BYTES)和真正从存储服务器上返回的数据量 (IO_INTERCONNECT_BYTES)。注意这些值是某语句所有执行的累加值。在整本书中我们都会用到这两列信息，因为这是卸载操作的关键指标。通过下面这个演示可以展示投影确实会影响返回到数据库服务器端的数据量，选择的列越少，传输的数据就越少。

```

SYS@SANDBOX> select /* single col */ avg(pk_col)
2 from kso.skew3;

```

AVG(PK_COL)

```

-----
16093750.3

```

1 row selected.

Elapsed: 00:00:32.13

```

SYS@SANDBOX> select /* multi col */ avg(pk_col),sum(col1)
2 from kso.skew3;

```

AVG(PK_COL) SUM(COL1)

```

-----
16093750.3 1.9003E+14

```

1 row selected.

■ 深入理解 Oracle Exadata

Elapsed: 00:00:45.32

```
SYS@SANDBOX> set timing off
```

```
SYS@SANDBOX> select sql_id,sql_text from v$sql
2 where sql_text like '%col */ avg(pk_col)%';
```

SQL_ID	SQL_TEXT
--------	----------

bb3z4aaa9du7j	select /* single col */ avg(pk_col) from kso.skew3
555pskb8aaqct	select /* multi col */ avg(pk_col),sum(col1) from kso.skew3

2 rows selected.

```
SYS@SANDBOX> select sql_id, IO_CELL_OFFLOAD_ELIGIBLE_BYTES eligible,
2 IO_INTERCONNECT_BYTES actual,
3 100*(IO_CELL_OFFLOAD_ELIGIBLE_BYTES-IO_INTERCONNECT_BYTES)
4 /IO_CELL_OFFLOAD_ELIGIBLE_BYTES "IO_SAVED_%", sql_text
5 from v$sql where sql_id in ('bb3z4aaa9du7j','555pskb8aaqct');
```

SQL_ID	ELIGIBLE	ACTUAL	IO_SAVED_%	SQL_TEXT
bb3z4aaa9du7j	1.6025E+10	4511552296	71.85	select /* single col */ avg(pk_col)
555pskb8aaqct	1.6025E+10	6421233960	59.93	select /* multi col */ avg(pk_col),s

2 rows selected.

```
SYS@SANDBOX> @fsx4
```

Enter value for sql_text: %col */ avg(pk_col)%

Enter value for sql_id:

SQL_ID	CHILD	OFFLOAD	IO_SAVED_%	AVG_ETIME	SQL_TEXT
6u7v77c2f8x5r	0	Yes	59.93	45.15	select /* multi col
d43dr7hvmw3yb	0	Yes	71.85	31.94	select /* single col

我们可以看到选择额外的列就会需要更多的时间来完成查询，V\$SQL 中的字段也印证了必须要传输的数据量有所增加。我们还显示了修改过的 fsx.sql 的输出结果，这在本章后面会详细讨论，当前你只需要知道它可以告诉我们一条语句有没有发生卸载。

2.2.2 谓词过滤

三大智能扫描优化方法中的第二个是谓词过滤，这是指 Exadata 能够只将感兴趣的行返回给数

数据库层。因为在 iDB 中，请求中包含了谓词信息，所以在返回数据前标准过滤操作在存储节点中就可以完成了。在使用非 Exadata 存储的数据库中，过滤是在数据库层完成的，这通常意味着大量最终会被丢弃的记录也要先返回到数据库层，而在存储层过滤这些行则会显著减少必须要传输到数据库层的数据量。当然这种优化也节省了数据库服务器上的 CPU 资源，不过最大的优势仍然是减少了数据量传输。

下面是一个例子。

```
SYS@SANDBOX> alter session set cell_offload_processing=false;
```

```
Session altered.
```

```
Elapsed: 00:00:00.01
```

```
SYS@SANDBOX> select count(pk_col) from kso.skew3;
```

```
COUNT(PK_COL)
```

```
-----
```

```
384000036
```

```
Elapsed: 00:00:48.45
```

```
SYS@SANDBOX> alter session set cell_offload_processing=true;
```

```
Session altered.
```

```
Elapsed: 00:00:00.01
```

```
SYS@SANDBOX> select count(pk_col) from kso.skew3;
```

```
COUNT(PK_COL)
```

```
-----
```

```
384000036
```

```
Elapsed: 00:00:26.61
```

```
SYS@SANDBOX> -- disable storage indexes
```

```
SYS@SANDBOX> alter system set "_kcfis_storageidx_disabled"=true;
```

```
System altered.
```

```
Elapsed: 00:00:00.17
```

```
SYS@SANDBOX> select count(pk_col) from kso.skew3 where col1 < 0;
```

```
COUNT(PK_COL)
```

```
-----
```

Elapsed: 00:00:08.53

首先我们通过设置 `CELL_OFFLOAD_PROCESSING` 来完全禁用卸载，然后运行一条没有 `WHERE` 子句的查询。没有卸载的加速，这条查询差不多耗时 48 秒。然后我们重新启用卸载，重新运行查询，这次仅仅大约耗时 27 秒。节省的 21 秒完全是字段投影的功劳（因为没有 `WHERE` 子句就没有其他的优化手段会发挥作用）。接下来我们通过设置隐含参数 `_KCFIS_STORAGEIDX_DISABLED=TRUE` 来禁用存储索引（下一小节我们会有更多讨论）并且添加了 `WHERE` 子句，这次执行时间缩短到 9 秒钟。这再次节省的 18 秒要归功于谓词过滤，请注意所有的性能提升都来自于谓词过滤，因为我们禁用了存储索引（以确保该语句不会从存储索引中获益），下面就来讨论一下“存储索引”。

2.2.3 存储索引

存储索引提供了智能扫描中第三个层次的优化。存储索引是一个在存储节点中的内存结构，维护了每个 1MB 磁盘存储单位中保存的最大值和最小值，每张表最多 8 列可以作为存储索引。存储索引跟其他大多数智能扫描优化手段略有不同，存储索引的目标不是为了减少传输回数据库层的数据量，实际上，对于一个查询，不管是不是使用到了存储索引，返回到数据库层的数据量都是一样的。存储索引的设计目的是为了缩减从存储服务器自身的磁盘中读取数据的时间，可以把这个功能想像成一个预先过滤。智能扫描将查询谓词发送到存储服务器，而存储索引中则包含了每个 1MB 存储区间的数值分布图，任何一个不可能包含符合条件的记录的区间都会被丢弃，不会被读取到。你还可以认为存储索引是另一种分区算法，磁盘 I/O 消减跟分区消减相类似，如果某个分区不会包含目标记录，那么该分区的数据块就不会被读取，同样的，如果一个存储区间不会包含目标记录，那么这个存储区间也无须被读取。

下面是一个例子。

```
SYS@SANDBOX> -- disable storage indexes
SYS@SANDBOX> alter system set "_kcfis_storageidx_disabled"=true;
```

System altered.

Elapsed: 00:00:00.22

```
SYS@SANDBOX> select count(pk_col) from kso.skew3 where col1 < 0;
```

```
COUNT(PK_COL)
```

```
-----
```

2

```
Elapsed: 00:00:08.74
SYS@SANDBOX> -- enable storage indexes
SYS@SANDBOX> alter system set "_kcfis_storageidx_disabled"=false;

System altered.
```

```
Elapsed: 00:00:00.03
SYS@SANDBOX> select count(pk_col) from kso.skew3 where col1 < 0;

COUNT(PK_COL)
-----
                2
```

```
Elapsed: 00:00:00.08
```

在上面的例子中，我们禁用了存储索引（通过设置_KCFIS_STORAGEIDX_DISABLED 参数）来展示使用字段投影和谓词过滤读取 3.84 亿条记录需要的时间，记住在本示例中，即使返回给数据库层的数据确实非常少，但是存储服务器还是要去读取 SKEW3 表的每一个包含数据的数据块，并且去检查每一行数据以确认是否满足 WHERE 条件，这就是这 8 秒钟所主要消耗的时间。然后我们重新启用了存储索引，重新运行了查询，执行时间缩短到.08 秒，这是因为使用了存储索引，避免了几乎所有的磁盘 I/O 以及从这些记录中过滤最终结果需要的时间。

重申一下，字段投影和谓词过滤（还有其他智能扫描的优化方式）通过减少传输回数据库服务器端的数据量来提高性能（同时减少传输时间），存储索引则是通过过滤、减少读取存储服务器中的数据块数量来提高性能。在第 4 章中我们将更详细地探讨存储索引。

2.2.4 简单连接（布隆过滤）

在某些情况下，连接操作也可以卸载到存储层去完成。卸载的连接操作是通过一种称为“布隆过滤（Bloom Filter）”的方式实现的。布隆过滤已经存在了很久，Oracle 是从 Oracle 数据库 10gR2 版本开始使用布隆过滤的，所以这并不是 Exadata 特有的功能。Oracle 使用它的一个主要用途是减少并行查询从属进程之间的通信。布隆过滤器的一个优势是相较于它们所要代表的数据而言，它们非常精小，但这是有代价的——它们有时会导致伪正确。具体地说，就是本来不应该在结果集中的行偶尔会穿越过滤器而跑到结果集中。因为这个原因，在运用完布隆过滤器之后，还要再使用另外一个过滤器来确保这些漏网之鱼被删除掉。从 Exadata 的角度来看，一件很有意思的事是，布隆过滤器可以交由存储层来做并在那里进行评估。这个技术可以极大地减少原本需要从存储上大量传输到数据库服务器上的数据量。

下面是一个例子：

```
SYS@SANDBOX> -- disable bloom filter offloading
```


■ 深入理解 Oracle Exadata

```
SYS@SANDBOX> alter session set "_bloom_predicate_pushdown_to_storage"=false;
```

Session altered.

Elapsed: 00:00:00.82

```
SYS@SANDBOX> @bloom_join2.sql
```

COL2	SUM(A.COL1)
2342	144
asddsadasd	153598416

2 rows selected.

Elapsed: 00:11:39.39

```
SYS@SANDBOX> -- enable bloom filter offloading
```

```
SYS@SANDBOX> alter session set "_bloom_predicate_pushdown_to_storage"=true;
```

Session altered.

Elapsed: 00:00:00.82

```
SYS@SANDBOX> @bloom_join2.sql
```

COL2	SUM(A.COL1)
asddsadasd	153598416
2342	144

2 rows selected.

Elapsed: 00:02:06.13

```
SYS@SANDBOX> @dplan
```

```
Enter value for sql_id: 09m6t5qpgkywx
```

```
Enter value for child_no: 0
```

PLAN_TABLE_OUTPUT

```
-----  
SQL_ID 09m6t5qpgkywx, child number 0  
-----
```

```
select /*+ bloom join 2 use_hash (skew temp_skew) */ a.col2,
```

```
sum(a.col1) from kso.skew3 a, kso.skew2 b where a.pk_col = b.pk_col and
b.col1 = 1 group by a.col2
```

Plan hash value: 466947137

Id	Operation	Name	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT				
1	PX COORDINATOR				
2	PX SEND QC (RANDOM)	:TQ10002	Q1,02	P->S	QC (RAND)
3	HASH GROUP BY		Q1,02	PCWP	
4	PX RECEIVE		Q1,02	PCWP	
5	PX SEND HASH	:TQ10001	Q1,01	P->P	HASH
6	HASH GROUP BY		Q1,01	PCWP	
* 7	HASH JOIN		Q1,01	PCWP	
8	BUFFER SORT		Q1,01	PCWC	
9	PX RECEIVE		Q1,01	PCWP	
10	PX SEND BROADCAST	:TQ10000		S->P	BROADCAST
11	TABLE ACCESS BY INDEX ROWID	SKEW2			
* 12	INDEX RANGE SCAN	SKEW2_COL1			
13	PX BLOCK ITERATOR		Q1,01	PCWC	
* 14	TABLE ACCESS STORAGE FULL	SKEW3	Q1,01	PCWP	

Predicate Information (identified by operation id):

```
7 - access("A"."PK_COL"="B"."PK_COL")
12 - access("B"."COL1"=1)
14 - storage(:Z>=:Z AND :Z<=:Z)
      filter(SYS_OP_BLOOM_FILTER(:BF0000,"A"."PK_COL"))
```

36 rows selected.

```
SYS@SANDBOX> @dplan
Enter value for sql_id: 09m6t5qpgkywx
Enter value for child_no: 1
```

PLAN_TABLE_OUTPUT

■ 深入理解 Oracle Exadata

SQL_ID 09m6t5qpgkywx, child number 1

```
select /*+ bloom join 2 use_hash (skew temp_skew) */ a.col2,
sum(a.col1) from kso.skew3 a, kso.skew2 b where a.pk_col = b.pk_col and
b.col1 = 1 group by a.col2
```

Plan hash value: 466947137

Id	Operation	Name	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT				
1	PX COORDINATOR				
2	PX SEND QC (RANDOM)	:TQ10002	Q1,02	P->S	QC (RAND)
3	HASH GROUP BY		Q1,02	PCWP	
4	PX RECEIVE		Q1,02	PCWP	
5	PX SEND HASH	:TQ10001	Q1,01	P->P	HASH
6	HASH GROUP BY		Q1,01	PCWP	
* 7	HASH JOIN		Q1,01	PCWP	
8	BUFFER SORT		Q1,01	PCWC	
9	PX RECEIVE		Q1,01	PCWP	
10	PX SEND BROADCAST	:TQ10000		S->P	BROADCAST
11	TABLE ACCESS BY INDEX ROWID	SKEW2			
* 12	INDEX RANGE SCAN	SKEW2_COL1			
13	PX BLOCK ITERATOR		Q1,01	PCWC	
* 14	TABLE ACCESS STORAGE FULL	SKEW3	Q1,01	PCWP	

Predicate Information (identified by operation id):

```
7 - access("A"."PK_COL"="B"."PK_COL")
12 - access("B"."COL1"=1)
14 - storage(:Z>=:Z AND :Z<=:Z AND SYS_OP_BLOOM_FILTER(:BF0000,"A"."PK_COL"))
    filter(SYS_OP_BLOOM_FILTER(:BF0000,"A"."PK_COL"))
```

36 rows selected.

在本例中，为了实现比较的效果，我们使用了一个隐含参数 `_BLOOM_PREDICATE_PUSHDOWN_TO_STORAGE` 来禁用卸载布隆过滤的功能。我们可以看到测试的语句在启用卸载时

运行了大约 2 分钟，而禁用时则运行了 11.5 分钟。如果再仔细观察执行计划中的谓词信息，则可以看到第二次运行时 `SYS_OP_BLOOM_FILTER(:BF0000,"A"."PK_COL")` 谓词是在存储节点中执行的。之所以启用卸载可以运行得更快，是因为存储节点可以对表进行预连接，这就消减了大量本来需要传输回数据库服务器的数据。

2.2.5 函数卸载

Oracle 实现的 SQL 语言中包含了很多内置的 SQL 函数，这些函数可以在 SQL 语句中直接使用，函数可以被分成两大类：单行函数和多行函数。对于查询的每张表中的每行记录，单行函数返回一条结果，这种单行函数又可以更进一步大致分为以下几种类型：

- 数字函数（`SIN`, `COS`, `FLOOR`, `MOD`, `LOG`, ...）
- 字符函数（`CHR`, `LPAD`, `REPLACE`, `TRIM`, `UPPER`, `LENGTH`, ...）
- 日期时间函数（`ADD_MONTHS`, `TO_CHAR`, `TRUNC`, ...）
- 转换函数（`CAST`, `HEXTORAW`, `TO_CHAR`, `TO_DATE`, ...）

几乎所有的这些单行函数都可以被卸载到 Exadata 存储上。

第二大类 SQL 函数操作的是多行记录，又分为两小类：

- 聚合函数（`AVG`, `COUNT`, `SUM`, ...）
- 分析函数（`AVG`, `COUNT`, `DENSE_RANK`, `LAG`, ...）

这些函数可能返回一行数据（聚合函数），也可能返回多行数据（分析函数），注意有些函数被重载之后会隶属于这两类。这些函数都不可以被卸载到 Exadata，这也合情合理，因为很多这类函数都要读取所有的记录集，而单个存储节点可能并不包含完整的数据（译者注：这是由于 ASM 的特性决定的，所有数据都被均衡到所有存储节点上，而 Exadata 的存储节点之间是 Share-nothing 架构）。

还有一些其他函数无法被完全归入上面所列的任何类别中，这些函数对于卸载而言各有不同，比如 `DECODE` 和 `NVL` 函数可以被卸载，而 `XML` 函数就不可以，某些数据挖掘的函数可以被卸载，而某些又不可以。需要注意的是，能够被卸载的函数列表会随着新版本的发布而有所不同。最终哪些函数具有卸载能力可以从 `V$SQLFN_METADATA` 视图中查询得知。

```
SYS@SANDBOX> select distinct name, version, offloadable
2 from v$sqlfn_metadata
3 order by 1,2;
```

NAME	VERSION	OFF
!=	SQL/DS	YES
!=	v6 Oracle	YES

■ 深入理解 Oracle Exadata

<	SQL/DS	YES
<	V6 Oracle	YES
<=	SQL/DS	YES
<=	V6 Oracle	YES
=	SQL/DS	YES
=	V6 Oracle	YES
>	SQL/DS	YES
>	V6 Oracle	YES
>=	SQL/DS	YES
>=	V6 Oracle	YES
ABS	V10 Oracle	YES
ABS	V10 Oracle	YES
ABS	V6 Oracle	YES
ACOS	V10 Oracle	YES
ACOS	V73 Oracle	YES
. . .		
VSIZE	V6 Oracle	YES
WIDTH_BUCKET	V82 Oracle	NO
XMLCAST	V11R1 Oracle	NO
XMLCDATA	V10 Oracle	NO
XMLCOMMENT	V10 Oracle	NO
XMLCONCAT	V92 Oracle	NO
XMLDIFF	V11R1 Oracle	NO
MLEXISTS2	V11R1 Oracle	NO
XMLISNODE	V92 Oracle	NO
XMLISVALID	V92 Oracle	NO
XMLPATCH	V11R1 Oracle	NO
XMLQUERY	V10 Oracle	NO
XMLTOOBJECT	V11R1 Oracle	NO
XMLTRANSFORM	V92 Oracle	NO
XMLTRANSFORMBLOB	V10 Oracle	NO
XS_SYS_CONTEXT	V11R1 Oracle	NO

921 rows selected.

可以卸载的函数实际上就是将通常由数据库服务器上 CPU 完成的工作交由存储节点来完成。不过，这对于节省 CPU 使用一般只有较小的提升，最大的收获还是来自于限制传回数据库服务器的数据量。通过评估 WHERE 子句中的函数，可以让存储节点仅仅将需要的行传输回数据库层，所以就跟其他的卸载操作一样，最主要的目的就是减少存储和数据库之间的数据流量。

2.2.6 压缩/解压缩

Exadata 中受到相当关注的一个功能是混合列式压缩（HCC），在智能扫描发生时，Exadata 将解压 HCC 压缩格式数据的工作卸载到存储节点中完成。也就是说，当智能扫描读取压缩数据的时候，需要的列是在存储节点中解压的。过滤操作并不需要解压，所以只有需要返回给数据库的数据才会被解压，不过要注意现在所有的压缩操作还都是在数据库层完成的，如果不使用智能扫描，解压缩也同样会在数据库层完成。简单起见，如表 2-1 所示。

表 2-1 HCC 压缩/解压缩卸载

操 作	数据库服务器	存储服务器
压缩	总是	从不
解压缩	非智能扫描	智能扫描

在存储层进行数据解压与其他大多数的智能扫描优化方式相比有些背道而驰，大多数方式是为了减少传输回数据库服务器的数据量（译者注：这里的意思是，其他的大多数方式都是为了尽量减少传输回数据库服务器的数据量，但是在存储层解压数据，然后再把解压完的数据传输回数据库恢复层，这无疑会增大传输的数据量，从这个层面上而言，是与其他方式背道而驰的），但是因为解压缩是一个十分消耗 CPU 的操作，特别是对于高压缩率的数据，所以还是决定将解压缩放到数据库服务器上完成。不过，这并不是一成不变的，在某些情况下，数据库服务器拥有足够的 CPU 资源，这样自行解压数据就成为颇具吸引力的选择（也就是在某些情况下减少数据传输量的效果超过了减少 CPU 占用的效果），事实上，在 11.2.2.3.1 cellsrv 版本中，存储节点繁忙的时候，Exadata 会选择将压缩数据直接返回给数据库服务器。

■ Kevin说：作者们正确地推测了数据库网格和存储网格之间的责任划分并非是一成不变的——至少从架构的视角来看。让我解释一下，在智能扫描过程中，Exadata存储服务器软件以 1MB数据块为单位执行卸载处理（如过滤、投影、解压EHCC数据），在真正的智能扫描模式下，智能扫描的结果（过滤和投影的数据）通过iDB返回到数据库节点中请求进程的PGA中，但是也总有很多情况，智能扫描不做如此处理而是以原块的形式返回数据。其中一个例子就是当智能扫描在过滤的时候遇到了行链接，只有数据库节点才能确定包含行链接的数据块的位置，即使存储节点足够智能到确认行链接的位置，那这个块也很可能不在同一个存储节点上，而存储节点之间又没有直接的通信路径。这个例子说明，但凡需要（needs to），智能扫描就能够恢复成块服务器（译者注：块服务器的意思是不做任何块处理而仅仅是将块直接返回，这就跟非Exadata的常规处理模式一样）。另外，只要想要（wants to），智能扫描也能够恢复成块服务器。我期待这本书在读者的书架上保留足够长的时间，长到能够让我们现在对Exadata数据库一体机中数据库节点和存储节点之

间角色和职责划分的某些死板描述不再有效。考虑一下这个场景，当存储节点处理一个对于深度压缩数据的最简单的并且拥有适度选择性的查询(比如，很少或根本没有表连接，以及很少的聚合和排序)，鉴于这些定义，在存储节点上过滤、投影、解压花费的CPU时间将远多于数据库节点进行数据处理(译者注：指的是前面说的连接、聚合、排序等处理)所花费的CPU时间(译者注：所以不如直接返回数据块)。我们知道智能扫描能够在需要 (needs to)的时候恢复成块服务器，我刚刚描述的这个场景则是智能扫描对于某些操作想要 (wants to)恢复成块服务器，如果有接近于空闲的处理器，当然不应该再把工作仍然分配到完全饱和的处理器上。

附带说明一下，有一个隐含参数可以用来控制解压操作是不是可以进行卸载。但是不幸的是，这个参数无法仅对解压操作生效(把解压操作放到存储节点上或是放到数据库节点上)，如果 `_CELL_OFFLOAD_HYBRIDCOLUMNAR` 参数被设置为 `FALSE`，那么在 HCC 数据上就完全禁用了智能扫描。

2.2.7 加密/解密

加密与解密操作跟压缩或者解压 HCC 数据的处理方法非常相近。加密永远在数据库层完成，而解密则可能在存储层也可能在数据库层完成。当通过智能扫描读取加密数据时，是在存储服务器上解密的，否则在数据库服务器上解密。X2-2 和 X2-8 平台在存储服务器上都是使用英特尔至强 Westmere 芯片(顺便提一下，X2-2 在数据库服务器上也用同样的芯片)，此芯片包含一个特殊的指令集(Intel AES-NI)可以对加密和解密进行有效的硬件加速，不过要注意，必须要 Oracle 数据库 11.2.0.2 版本才能够获得新指令集带来的好处。

加密和 HCC 压缩可以一起工作，由于压缩是首先完成的，所以加密和解密 HCC 数据的工作量会少一些。通过 `CELL_OFFLOAD_DECRYPTION` 参数可以控制加密解密的行为，不过就跟隐含参数 `_CELL_OFFLOAD_HYBRIDCOLUMNAR` 一样，设置为 `FALSE` 就在加密数据上完全禁用了智能扫描，也同时禁用了在存储层的解密。

2.2.8 虚拟列

虚拟列可以用来定义从表中的其他字段计算而得的伪列，不用真正存储这些计算后的值。虚拟列可以被用于分区键、约束、索引，同样也可以收集列级统计信息，由于虚拟列的值并不是真正存储的，所以在被读取到的时候必须实时计算，而当通过智能扫描读取时，这些计算则可以被卸载到存储层。

```
SYS@SANDBOX1> alter table kso.temp_skew add col1_plus_pk as (col1+pk_col);
```


Table altered.

```
SYS@SANDBOX1> select col1_plus_pk from kso.temp_skew where rownum < 10;
```

COL1_PLUS_PK

```
-----
27998260
27998258
27998256
27998254
27998252
27998250
27998248
27998246
27998244
```

9 rows selected.

```
SYS@SANDBOX1> select count(*) from kso.temp_skew where col1_plus_pk=27998244;
```

```
COUNT(*)
-----
2
```

```
SYS@SANDBOX> @fsx4
```

Enter value for sql_text: select count(*) from kso.temp_skew where col1_plus_pk=27998244

Enter value for sql_id:

SQL_ID	CHILD	OFFLOAD	AVG_ETIME	IO_SAVED_%	SQL_TEXT
35tqjjq5vzg4b	0	Yes	1.14	99.99	select count(*) from

1 row selected.

```
SYS@SANDBOX1> @dplan
```

Enter value for sql_id: 35tqjjq5vzg4b

Enter value for child_no:

PLAN_TABLE_OUTPUT

```
-----
```

■ 深入理解 Oracle Exadata

SQL_ID 35tqjjq5vzg4b, child number 0

select count(*) from kso.temp_skew where col1_plus_pk=27998244

Plan hash value: 725706675

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	

0	SELECT STATEMENT				44804 (100)		
1	SORT AGGREGATE		1	13			
* 2	TABLE ACCESS STORAGE FULL	TEMP_SKEW	320K	4062K	44804 (2)	00:08:58	

Predicate Information (identified by operation id):

2 - storage("COL1"+"PK_COL"=27998244)
filter("COL1"+"PK_COL"=27998244)

20 rows selected.

SYS@SANDBOX1> alter session set "_cell_offload_virtual_columns"=false;

Session altered.

SYS@SANDBOX1> @flush_sql

Enter value for sql_id: 35tqjjq5vzg4b

PL/SQL procedure successfully completed.

SYS@SANDBOX1> select count(*) from kso.temp_skew where col1_plus_pk=27998244;

COUNT(*)

2

1 row selected.

SYS@SANDBOX1> @fsx4

```

Enter value for sql_text: select count(*) from kso.temp_skew where col1_plus_pk=27998244
Enter value for sql_id:
Enter value for inst_id:

```

SQL_ID	CHILD	OFFLOAD	AVG_ETIME	IO_SAVED_%	SQL_TEXT
35tqjjq5vzg4b	0	Yes	3.00	59.79	select count(*) from

1 row selected.

```
SYS@SANDBOX1> alter session set "_cell_offload_virtual_columns"=true;
```

Session altered.

```
SYS@SANDBOX1> select count(*) from kso.temp_skew where col1_plus_pk=27998244;
```

COUNT(*)
2

1 row selected.

```
SYS@SANDBOX1> @fsx4
```

```

Enter value for sql_text: select count(*) from kso.temp_skew where col1_plus_pk=27998244
Enter value for sql_id:
Enter value for inst_id:

```

SQL_ID	CHILD	OFFLOAD	AVG_ETIME	IO_SAVED_%	SQL_TEXT
35tqjjq5vzg4b	0	Yes	3.00	59.79	select count(*) from
35tqjjq5vzg4b	0	Yes	1.19	99.99	select count(*) from

2 rows selected.

这个例子演示了虚拟列计算是可以被卸载的,而且也展示了通过 `_CELL_OFFLOAD_VIRTUAL_COLUMNS` 参数来控制优化器,注意,存储索引是不会创建在虚拟列上的。跟函数卸载一样,卸载虚拟列计算真正的长处在于减少了返回给数据库服务器的数据量,这比减少数据库服务器的 CPU 消耗要重要。

2.2.9 数据挖掘模型评分

一些数据模型评分函数是可以被卸载的,通常我们说这种优化也是为了减少传输回数据库层的数据量,而不仅仅是为了纯粹的 CPU 减负。跟其他的函数卸载一样,可以通过查询 V\$SQLFN_METADATA 视图来确认哪些数据挖掘函数是可以被卸载的,输出如下所示:

```
SYS@SANDBOX> select distinct name, version, offloadable
2 from V$SQLFN_METADATA
3 where name like 'PREDICT%'
4 order by 1,2;
```

NAME	VERSION	OFFLOADABLE
PREDICTION	V10R2 Oracle	YES
PREDICTION_BOUNDS	V11R1 Oracle	NO
PREDICTION_COST	V10R2 Oracle	YES
PREDICTION_DETAILS	V10R2 Oracle	NO
PREDICTION_PROBABILITY	V10R2 Oracle	YES
PREDICTION_SET	V10R2 Oracle	NO

6 rows selected.

如上所示,有些函数可以卸载,有些却不可以。能够卸载的函数可以被用来作为存储节点上的谓词过滤。这里是一个例子,只会返回那些符合在 WHERE 子句中定义的评分需求的记录:

```
select cust_id
from customers
where region = 'US'
and prediction_probability(churnmod,'Y' using *) > 0.8;
```

这种优化的设计目的是为了在减少数据传输同时也卸载 CPU 使用,不过就如上面的例子所示,减少传输回数据库层的数据量是最有收益的。

2.2.10 非智能扫描类型的卸载

还有一些优化与查询处理无关,由于并非本章重点,所以我们仅做简要描述。

1. 智能文件创建

这种优化在名称上会有些误导,它实际上是为了加速数据块初始化动作的。只要有数据块分配,数据库就必须初始化这些块。这个动作会发生在创建表空间的时候,也会发生在由于各种原因而产生的文件扩展时。在非 Exadata 存储上,这种情况就需要数据库服务器格式化每个数据块,然后再

将它们写回磁盘，所有的这些读写操作都会导致数据库服务器和存储之间大量的数据流量，正如你现在知道的，消除层与层之间数据流量是 Exadata 的主要目标，所以如你所猜，这些完全没有必要的流量就被消除了。数据块在存储节点上被格式化，而无须发送到数据库服务器中来。该动作所等待的时间由 Smart File Creation 等待事件记录。这个等待事件和调用它的操作将在第 10 章详细描述。

2. RMAN 增量备份

Exadata 通过增加数据块变化跟踪（block change tracking）粒度来加速增量备份。在非 Exadata 平台上，块变化是通过一组块来跟踪的；在 Exadata 中，则是对单个块跟踪变化。这样就显著减少了必须备份的数据块个数，结果是更小的备份尺寸、更少的 I/O 带宽、更短的增量备份时间。此功能能够通过设置 `_DISABLE_CELL_OPTIMIZED_BACKUPS=TRUE` 来禁用。我们将在第 9 章中讲述此优化。

3. RMAN 恢复（Restore）

此优化加快了从存储节点恢复备份时文件初始化部分的速度，就算很少会从备份中恢复数据库，此优化也能够帮助加速克隆环境。此优化减少了数据库服务器的 CPU 使用率，减少了两层之间的数据传输量。设置 `_CELL_FAST_FILE_RESTORE=FALSE`，将禁用该行为。我们仍将在第 9 章中讲述此优化。

2.3 智能扫描的先决条件

智能扫描不会发生在 Exadata 中的每一个查询中，如果要发生，那么必须满足三个基本条件：

- 必须要是对象上的全扫描。
- 扫描必须要使用 Oracle 直接路径读取机制。
- 对象必须存储在 Exadata 存储中。

简单解释一下为什么存在这些要求。Oracle 是一个 C 程序，执行智能扫描的函数（`kcfis_read`）是由直接路径读取函数（`kcblldrget`）调用的，就是这么简单，不通过全扫描到直接读取这条代码路径，就无法使用 `kcfis_read` 函数，当然，存储上也必须要运行 Oracle 软件才可以处理智能扫描。

接下来我们将依次讨论这些条件。

2.3.1 全扫描

为了查询可以获得 Exadata 卸载能力的优势，优化器必须决定使用全表扫描或者快速全索引扫描来执行语句，这里的用词比较笼统，一般来说，这两个词对应的是执行计划中的 `TABLE ACCESS FULL` 和 `INDEX FAST FULL SCAN`。在 Exadata 中，这些类似的操作的命名做了些许更改以表明访问的是 Exadata 存储。新的操作名称是 `TABLE ACCESS STORAGE FULL` 和 `INDEX STORAGE FAST FULL SCAN`。请注意，也有一些细微的变化，比如 `MAT_VIEW ACCESS STORAGE FULL`

事件也表示可以使用智能扫描。不过你应该知道，事实上就算执行计划里面显示 TABLE ACCESS STORAGE FULL 操作，也并不意味着查询就一定执行了智能扫描，它仅仅意味着前提已经满足。我们将在本章稍后探讨如何确认一个语句是否确实通过智能扫描实现了卸载操作。

2.3.2 直接路径读取

智能扫描除了必须是全扫描之外，还需要读取操作是通过 Oracle 直接路径读取机制来执行的。直接路径读取已经存在很长一段时间了，传统上，这种读取机制是被服务于并行查询的从属进程（Slave Process）使用的。因为并行查询最初预计将用于访问非常大量的数据（通常大到无法全部放入 Oracle 缓冲区），所以决定并行从属进程直接读取数据，然后放入自己的内存中（也被称为程序全局区或者 PGA）。直接路径读取机制完全跳过了标准的将数据库放入缓冲区这样的 Oracle 缓存机制，这对海量数据是非常有效的，因为它消除了那些额外的没有帮助的工作（缓存全表扫描获取到的却又可能不会被再次用到的数据），让这些块不至于将其他的数据块刷新出缓冲区。像我们之前提到的，kcfs（Kernel File Intelligent Storage）函数是被 kcbldrget（Kernel Block Direct Read GET）函数调用的，所以，智能扫描只有在使用直接路径读取机制的时候才会执行。

除了并行从属进程，只要条件允许，直接路径读取也可能用在非并行 SQL 语句中。有一个隐含参数 SERIAL_DIRECT_READ 可以控制此功能。当此参数设置为默认值 AUTO 时，Oracle 自动判断是否要为非并行扫描使用直接路径读取。计算基于几个因素，包括对象大小、Buffer Cache 大小，以及在 Buffer Cache 中已存在了多少该对象的数据块。另外，还有一个隐含参数（_SMALL_TABLE_THRESHOLD）定义了如果要使用串行直接路径读取，那么表至少要为多大。对于非串行扫描决定是否要使用直接路径读取机制的算法并未公开，虽然串行直接路径读取的功能早已存在，但是只是在最近才成为比较普遍的现象。Oracle 数据库 11gR2 在计算是否对于非并行扫描使用直接路径读取上做了一些修改，新修改的算法使 Oracle 11gR2 比以前的数据库版本要更频繁地采用直接路径读取。这也许是因为有了 Exadata 智能扫描，因此希望尽可能地触发直接路径读取，但是这样的算法在非 Exadata 平台上可能略显激进。

■ 备注 My Oracle Support 文档：793845.1 中包含如下表述。

在 11g 中，关于在串行表扫描中是使用直接路径读取还是使用缓存读取，我们做了探索性的改动。在 10g 中，对于大表的串行扫描默认是通过缓存的，在 11g 中，决定是直接读取还是通过缓存读取，要基于表大小、缓冲区大小和其他多种统计信息。因为避免了闕锁（Latch），因此直接路径读取比离散读（Scattered Read）更快，对其他进程影响更小。

2.3.3 Exadata 存储

要想智能扫描发生，当然扫描的数据都要保存在 Exadata 存储中。虽然在 Exadata 数据库服务器中也可以创建访问非 Exadata 存储的 ASM 磁盘组，不过显而易见，任何访问了这些非 Exadata 磁盘组中对象的 SQL 语句都无法进行卸载。虽然不常见，但是创建混合了 Exadata 存储和非 Exadata 存储的 ASM 磁盘组的情况也还是有的，比如通过 ASM Rebalance 方式来完成迁移的工作。对这些创建在混合存储上的磁盘组中的对象进行检索是无法用到智能扫描的。实际上 ASM 磁盘组中有一个属性（cell.smart_scan_capable）定义了一个磁盘组是否能够进行智能扫描，在分配非 Exadata 存储到一个 ASM 磁盘组以前，这个属性必须设置为 FALSE。下面是创建混合型存储磁盘组的过程，并且演示了检索存储在该磁盘组中的表的效果。该例对读者而言可能有些超前，所以其中一些命令读者可能还不知道是什么意思，不过别着急，我们会在合适的时间进行说明。现在，我们只希望你看到，可以访问非 Exadata 存储，但是那将会禁用所有基于 Exadata 存储的优化方式。

```
SYS@+ASM> -- Add non-Exadata storage
```

```
SYS@+ASM> alter diskgroup SMITHERS add failgroup LOCAL disk '/dev/raw/raw5','/dev/raw/raw6';
alter diskgroup SMITHERS add failgroup LOCAL disk '/dev/raw/raw5','/dev/raw/raw6'
*
```

```
ERROR at line 1:
```

```
ORA-15032: not all alterations performed
```

```
ORA-15285: disk '/dev/raw/raw5' violates disk group attribute cell.smart_scan_capable
```

```
ORA-15285: disk '/dev/raw/raw6' violates disk group attribute cell.smart_scan_capable
```

```
SYS@+ASM> alter diskgroup smithers set attribute 'cell.smart_scan_capable' = 'FALSE';
```

```
Diskgroup altered.
```

```
SYS@+ASM> alter diskgroup SMITHERS add failgroup LOCAL disk '/dev/raw/raw5','/dev/raw/raw6';
```

```
Diskgroup altered.
```

```
SYS@+ASM> select name, total_mb from v$asm_diskgroup where state='MOUNTED'
```

NAME	TOTAL_MB
SMITHERS	512,000
SMITHERS_LOCAL	1,562

```
SYS@+ASM> select g.name "diskgroup", d.path "disk", d.failgroup "failgroup", d.total_mb "disk
size" from v$asm_diskgroup g, v$asm_disk d where g.group_number=d.group_number and
```


■ 深入理解 Oracle Exadata

```
g.state='MOUNTED'
```

diskgroup	disk	failgroup	disk size
SMITHERS	/dev/raw/raw5	LOCAL	102,400
SMITHERS	/dev/raw/raw6	LOCAL	102,400
SMITHERS	o/192.168.12.3/SMITHERS_CD_05_cell01	ENKCEL01	102,400
SMITHERS	o/192.168.12.4/SMITHERS_CD_05_cell02	ENKCEL02	102,400
SMITHERS	o/192.168.12.5/SMITHERS_CD_05_cell03	ENKCEL03	102,400
SMITHERS_LOCAL	/dev/raw/raw1	SMITHERS_LOCAL_0000	781
SMITHERS_LOCAL	/dev/raw/raw2	SMITHERS_LOCAL_0001	781

7 rows selected.

我们有一个包含了 HCC 压缩表的在 Exadata 存储上的磁盘组，一开始我们尝试往这个磁盘组中添加非 Exadata 存储，错误显示我们违反了 `cell.smart_scan_capable` 磁盘组属性，当我们修改了属性，将之改为 `FALSE` 以后，就可以添加非 Exadata 存储了（当然，修改这个设定以后就禁用了所有存储在此磁盘组中的对象上的智能扫描），然后我们登录数据库，尝试访问我们的压缩表：

```
SYS@SMITHERS> select table_name, compression, compress_for
  2 from dba_tables where owner='ACOLVIN';
```

TABLE_NAME	COMPRESS	COMPRESS_FOR
SKEW3	ENABLED	QUERY HIGH

```
SYS@SMITHERS> @table_size
```

```
Enter value for owner: ACOLVIN
Enter value for table_name: SKEW3
Enter value for type: TABLE
```

OWNER	SEGMENT_NAME	TYPE	TOTALSIZE_MEGS TS
ACOLVIN	SKEW3	TABLE	1,020.0 USERS
sum			1,020.0

```
SYS@SMITHERS> select count(*) from acolvin.skew3 where col1<0;
select count(*) from acolvin.skew3 where col1<0
```

*

ERROR at line 1:

ORA-64307: hybrid columnar compression is only supported in tablespaces residing on Exadata storage

```
SYS@SMITHERS> alter table acolvin.skew3 move nocompress;
```

```
Table altered.
```

```
SYS@SMITHERS> select /*+ parallel (a 8) */ count(*) from acolvin.skew3 a;
```

```

COUNT(*)
-----
384000048

```

```
Elapsed: 00:03:24.64
```

```
SYS@SMITHERS> @fsx4
```

```
Enter value for sql_text: select /*+ parallel (a 8) */ count(*) from acolvin.skew3 a
```

```
Enter value for sql_id:
```

SQL_ID	CHILD	OFFLOAD	IO_SAVED_%	AVG_ETIME	SQL_TEXT
5y9jm9pfbrg7q	0	No	.00	204.58	select /*+ parallel

```
1 row selected.
```

```
SYS@SMITHERS> @table_size
```

```
Enter value for owner: ACOLVIN
```

```
Enter value for table_name: SKEW3
```

```
Enter value for type: TABLE
```

OWNER	SEGMENT_NAME	TYPE	TOTALSIZE_MEGS	TS
ACOLVIN	SKEW3	TABLE	13,814.0	USERS
sum			13,814.0	

```
1 row selected.
```

所以，添加了非 Exadata 存储同时也就禁用了 HCC，我们为了访问表则必须先解压缩，当解压完成我们在表上执行了并行查询，如你所见，查询并未被卸载。所有这些都表明如果想要执行智能扫描就必须要求 Exadata 存储。

2.4 无法使用智能扫描的情况

有些情况下智能扫描会被有效禁用，最简单的情况就是在 Exadata 代码中还没有实现该功能，因此智能扫描根本就不会发生。还有些情况，Oracle 发起智能扫描，但是存储软件决定（或者是被迫），恢复到块运输方式，通常这是以块为单位来决定的（译者注：原文作者这里用的词是 block-by-block，也就是说在一次查询中，可能某些块使用了智能扫描而另一些块却无法使用，详见后面的 2.4.2 节“转换为块运输模式”）。

2.4.1 未实现的功能

在智能扫描优化的讨论中，我们已经描述了启用智能扫描必须满足的先决条件。然而，即使这些条件得到满足，还是有智能扫描无法发生的情况。在以下这些情况中，智能扫描就根本无法使用，这与如何优化无关（至少在 cellsrv 版本 11.2.2.2.0 中是如此）。

- 聚簇表中无法使用智能扫描。
- 索引组织表中无法使用智能扫描。
- 在启用了 ROWDEPENDENCIES 的表中无法使用智能扫描。

2.4.2 转换为块运输模式

有些情况下，当智能扫描被启用，但是却由于某些原因 cellsrv 恢复成了块运输模式。这是一个相当复杂的话题，我们有些纠结是否需要将它包含在介绍卸载的章节中，但它是一个基本概念，因此最后我们还是决定在这里探讨一下。

我们已经描述过智能扫描是怎样通过将预过滤的数据直接返回 PGA 从而避免传输大量数据到数据库层，那么这里要理解的关键概念就是智能扫描也可以选择（或被迫）将完整的数据块返回 SGA。基本上，任何会导致 Oracle 为了获取完整的记录而要去读取另一个块的情况都会导致这种情况发生。行链接大概是最简单的例子。当 Oracle 遇到一个链接的行，在行中会保存一个指针指向保存着此记录第二部分的数据块，由于存储节点互相之间并无直接通信，并且链接的块很可能没有保存在同一个存储节点上，因此 cellsrv 就索性返回整个数据块，交由数据库层来处理。

所以在这个非常简单的例子里，智能扫描会暂停片刻，执行一次单块读，再推动另一次单块读来获取额外的记录信息。请记住这仅仅是一个非常简单的例子。同样的行为也会发生在当 Oracle 必须要处理读一致性问题的时候，比如说，如果 Oracle 意识到数据块比现在正在进行的查询要新，那么找到正确的前镜像版本数据块的工作就会交给数据库层来处理，这样智能扫描过程也会被暂停，数据库会执行传统的读一致性操作。

那么这真的很重要吗？为什么你应该关注呢？答案自然是“it depends”——这要看情况。在大多数情况下，你不需要关心，即使执行了智能扫描，Oracle 也会保证读的一致性，这是很重要的

一点。事实上 Oracle 的行为跟应用的立场是完全一样的，它不会认为是否使用智能扫描是一个大问题，Oracle 会连同智能扫描一起做一些单块读，而不会过多考虑结果是否正确、性能是否受到很大影响，大多数情况下都不会。但是也有一些情况，选择执行智能扫描后又恢复到块传输模式，会对性能产生巨大影响，这些例子就需要你去关注并且理解后台到底发生了什么。在第 16 章中读者能找到更多信息。

2.4.3 跳过某些卸载操作

我们再简要描述一下另一个同样很复杂的行为，`cellsrv` 可能会拒绝执行某些普通的卸载操作，比如可能是为了避免存储节点上的 CPU 资源超载。这种行为的一个很好的例子就是解压 HCC 数据。解压缩是一个非常消耗 CPU 资源的任务，特别是对于那些高压缩率的数据。在 Exadata 存储软件的后续版本中（11.2.2.3.0 或者更新），当存储节点上的 CPU 非常繁忙而数据库节点中的 CPU 却不太忙时，`cellsrv` 能够选择跳过解压一部分数据的步骤。通过强制数据库节点进行解压操作，有效地将负载转移回了数据库层。在这种情况下，一些步骤，比如投影还是会在存储节点中完成，而不论解压缩步骤是不是已经被跳过了。正如 Kevin 已经指出的那样，这种可以选择如何执行 offload 操作的能力会使各层之间的边界并不明确而变得难以监控。在第 12 章中会探讨一些技术来监控这些行为。

2.5 如何验证智能扫描确实发生

想要了解 Exadata，很重要的一点就是如何去确认查询是不是从智能扫描中获益。这并不像听上去那么简单，很不幸，通过 `DBMS_XPLAN` 包输出的普通执行计划不会显示智能扫描是否被使用了。下面是一个例子：

PLAN_TABLE_OUTPUT

SQL_ID 05cq2hb1r37tr, child number 0

select avg(pk_col) from kso.skew a where col1 > 0

Plan hash value: 568322376

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				44486 (100)	
1	SORT AGGREGATE		1	11		
* 2	TABLE ACCESS STORAGE FULL	SKEW	32M	335M	44486 (1)	00:08:54

Predicate Information (identified by operation id):

```
-----  
2 - storage("COL1">0)  
   filter("COL1">0)
```

我们可以看到优化器选择了 TABLE ACCESS STORAGE FULL 操作，并且在执行计划第 2 步的谓词部分显示了 storage()谓词，这两个特征都表示智能扫描是可能发生的，但是却都没有提供明确的验证。实际上，本例中的这个语句并没有执行智能扫描。

■ 备注 执行计划输出中有一个有趣的功能值得一提。我们注意到在谓词部分有一个 storage()子句，还有一个进行相同比较的 filter()子句。我们对此着实疑惑了一阵儿，这两个子句到底是代表着独立的执行计划还是仅仅只是一种奇怪的 XPLAN 输出？实际上解释这个行为很简单。存储节点在很多情况下都必须恢复成常规的数据块 I/O 模式，如在智能扫描时遇到了行链接，该块必须整体返回给数据库层。正是由于不可能保证“纯粹的”智能扫描，所以执行计划中除了存储层的过滤（指的是 storage()子句）之外还必须包含 Filter 过滤操作。所以这两个谓词实际上指的是两种不同的操作，但是请记住，这并不会重复操作，filter()操作将在块传输模式的时候完成记录的返回，而 storage()操作则在存储节点上执行，通过正常的智能扫描算法将记录直接返回数据库 PGA 中。

执行计划不能显示是否进行了智能扫描，这确实有些让人沮丧，但是，我们可以用一些其他技术来解决这个问题。接下来的几节我们将介绍几种方法。

2.5.1 10046 跟踪

最直接的确认智能扫描是否执行的方法就是在有疑问的 SQL 语句上打开 10046 跟踪，不过，这种方法比较麻烦，而且还无法调查在之前的执行中发生了什么，尽管如此，跟踪还是一个用来确认智能扫描是否被执行的万无一失的方法。如果使用了智能扫描，那么在跟踪文件中就会有 CELL SMART TABLE SCAN 或者 CELL SMART INDEX SCAN 等待事件，下面是从上条语句的跟踪文件中摘录的。

```
PARSING IN CURSOR #47387827351064 len=49 dep=0 uid=0 oct=3 lid=0 tim=1297219338278533
  hv=3279003447 ad='2c8743808' sqlid='05cq2hb1r37tr'
select avg(pk_col) from kso.skew a where col1 > 0
END OF STMT
PARSE #47387827351064:c=57991,e=78256,p=25,cr=199,cu=0,mis=1,r=0,dep=0,
og=1,plh=568322376,tim=12
EXEC #47387827351064:c=0,e=14,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=568322376,tim=1297
WAIT #47387827351064: nam='SQL*Net message to client' ela= 2 . . .
WAIT #47387827351064: nam='cell single block physical read' ela= 487 . . .
WAIT #47387827351064: nam='cell multiblock physical read' ela= 25262 . . .

*** 2011-02-08 20:42:19.106
WAIT #47387827351064: nam='cell multiblock physical read' ela= 20303 . . .
WAIT #47387827351064: nam='gc cr multi block request' ela= 493 . . .
WAIT #47387827351064: nam='gc cr multi block request' ela= 271 . . .
WAIT #47387827351064: nam='cell multiblock physical read' ela= 2550 . . .
*** 2011-02-08 20:42:20.107
WAIT #47387827351064: nam='cell multiblock physical read' ela= 3095 . . .
WAIT #47387827351064: nam='gc cr multi block request' ela= 548 . . .
WAIT #47387827351064: nam='gc cr multi block request' ela= 331 . . .
WAIT #47387827351064: nam='cell multiblock physical read' ela= 22930 . . .
```

我们可以看到在跟踪文件中并没有智能扫描的等待事件。下面给出一个使用了智能扫描的语句的跟踪文件来进行比较。

```
PARSING IN CURSOR #2 len=32 dep=0 uid=0 oct=3 lid=0 hv=123 ad='196' sqlid='162wjnvwybhn'
select sum(pk_col) from kso.skew
END OF STMT
PARSE #2:c=2000,e=2424,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,plh=568322376
EXEC #2:c=0,e=34,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=568322376
WAIT #2: nam='SQL*Net message to client' ela= 3 driver id=1650815232 #bytes=1 p3=0 obj#=-1
WAIT #2: nam='ges message buffer allocation' ela= 2 pool=0 request=1 allocated=0 obj#=-1
WAIT #2: nam='KJC: wait for msg sends to complete' ela= 10 msg=6674450368 dest|rcvr=65536
```

```

WAIT #2: nam='reliable message' ela= 1107 channel context=6712270872 channel handle=66967991
WAIT #2: nam='ges message buffer allocation' ela= 1 pool=0 request=1 allocated=0 obj#=-1
WAIT #2: nam='enq: KO - fast object checkpoint' ela= 104 name|mode=126 2=65575 0=1 obj#=-1
WAIT #2: nam='ges message buffer allocation' ela= 1 pool=0 request=1 allocated=0 obj#=-1
WAIT #2: nam='enq: KO - fast object checkpoint' ela= 103 name|mode=126 2=65575 0=2 obj#=-1
WAIT #2: nam='cell smart table scan' ela= 162 cellhash#=2133459483 p2=0 p3=0 obj#=66849
WAIT #2: nam='cell smart table scan' ela= 244 cellhash#=379339958 p2=0 p3=0 obj#=66849
WAIT #2: nam='cell smart table scan' ela= 181 cellhash#=3176594409 p2=0 p3=0 obj#=66849
WAIT #2: nam='cell smart table scan' ela= 1285 cellhash#=2133459483 p2=0 p3=0 obj#=66849
WAIT #2: nam='cell smart table scan' ela= 1327 cellhash#=379339958 p2=0 p3=0 obj#=66849
WAIT #2: nam='cell smart table scan' ela= 1310 cellhash#=3176594409 p2=0 p3=0 obj#=66849
WAIT #2: nam='cell smart table scan' ela= 19755 cellhash#=3176594409 p2=0 p3=0 obj#=66849
WAIT #2: nam='cell smart table scan' ela= 39 cellhash#=3176594409 p2=0 p3=0 obj#=66849

```

正如你看到的，这个跟踪文件中有不少 CELL SMART TABLE SCAN 等待事件，因此毫无疑问，此语句执行了智能扫描。我们会在第 10 章更加详细地探讨 Exadata 特有的和与 Exadata 相关的等待事件。

2.5.2 性能统计 (v\$sesstat)

当然，我们也可以查看一些性能视图，比如 V\$SESSTAT 和 V\$ACTIVE_SESSION_HISTORY。Tanel Poder 的 Snapper 脚本提供了一个很好的方式去查看语句在执行时都产生了什么等待事件。但是，需要在语句正在执行的时候才能抓到发生了什么。活动会话历史（ASH）也同样不错，但是因为数据是采样的，所以并不保证你一定可以抓到你想找到的等待事件。即便如此，只要你能在语句执行的时候访问系统，性能统计都能提供一个可靠的数据来源。下面是一个使用 V\$MYSTATS 视图的例子，是一个只显示你当前会话数据的 V\$SESSTAT。在这个例子中我们能查看存储节点扫描的统计信息，智能扫描发生的话，这个统计值就会增加。

```

SYS@dbm1> set echo on
SYS@dbm1> @mystats
SYS@dbm1> select name, value
2 from v$mystat s, v$statname n
3 where n.statistic# = s.statistic#
4 and name like nvl('%&name%',name)
5 order by 1
6 /

```

Enter value for name: cell scans

NAME	VALUE

cell scans

833

1 row selected.

SYS@dbm1> set echo off

SYS@dbm1> select avg(pk_col) from kso.skew2 a where col1 > 0;

AVG(PK_COL)

16093748.8

1 row selected.

SYS@dbm1> @mystats

Enter value for name: cell scan

NAME

VALUE

-----	-----
cell scans	834

1 row selected.

SYS@dbm1> alter session set cell_offload_processing=false;

Session altered.

SYS@dbm1> select avg(pk_col) from kso.skew2 a where col1 > 0;

AVG(PK_COL)

16093748.8

1 row selected.

SYS@dbm1> @mystats

Enter value for name: cell scans

NAME

VALUE

-----	-----
cell scans	834

1 row selected.

正如你所见，第一次执行语句的时候有卸载操作，存储节点扫描的统计值自 833 增加到了 834，然后我们关闭了智能扫描又再次运行了语句，这次统计值并没有增加。因此，只要我们能够实时地抓到这条语句，这种方法的效果就会很好。请注意，Oracle 性能统计与 Oracle 等待事件接口相辅相成，提供了其他地方都无法找到的信息。我们将在第 11 章详细讨论 Exadata 相关的统计信息。

2.5.3 卸载适用字节

还有另外一条线索也可以显示语句是否使用了智能扫描。我们之前已经提过，V\$SQL 视图包含了一列，名为 IO_CELL_OFFLOAD_ELIGIBLE_BYTES，显示了适合进行卸载的数据字节数。这列可以作为表示智能扫描是否被使用的指标。只有在智能扫描被使用的时候，这列值才会大于 0。通过这个发现，我们写了一个小脚本（fsx.sql），基于判断 V\$SQL 中的该列值是否大于 0 来返回 YES 或 NO，脚本的输出较宽，不适合放在本书中，所以我们在例子中使用了一些精简版的脚本，当然，所有版本的脚本都可以从在线代码库中获得。你也许在前面几个小节中就已经看过这个代码的执行效果了，下面是该脚本的内容和使用范例。

```
SYS@SANDBOX1> !cat fsx.sql
-----
--
-- File name: fsx.sql
--
-- Purpose: Find SQL and report whether it was Offloaded and % of I/O saved.
--
-- Usage: This scripts prompts for two values.
--
-- sql_text: a piece of a SQL statement like %select col1, col2 from skew%
--
-- sql_id: the sql_id of the statement if you know it (leave blank to ignore)
--
-- Description:
--
-- This script can be used to locate statements in the shared pool and
-- determine whether they have been executed via Smart Scans.
--
-- It is based on the observation that the IO_CELL_OFFLOAD_ELIGIBLE_BYTES
-- column in V$SQL is only greater than 0 when a statement is executed
-- using a Smart Scan. The IO_SAVED_% column attempts to show the ratio of
-- of data received from the storage cells to the actual amount of data
-- that would have had to be retrieved on non-Exadata storage. Note that
```

```

--          as of 11.2.0.2, there are issues calculating this value with some queries.
--
--          Note that the AVG_ETIME will not be accurate for parallel queries. The
--          ELAPSED_TIME column contains the sum of all parallel slaves. So the
--          script divides the value by the number of PX slaves used which gives an
--          approximation.
--
--          Note also that if parallel slaves are spread across multiple nodes on
--          a RAC database the PX_SERVERS_EXECUTIONS column will not be set.
--
-----
set pagesize 999
set lines 190
col sql_text format a70 trunc
col child format 99999
col execs format 9,999
col avg_etime format 99,999.99
col "IO_SAVED%" format 999.99
col avg_px format 999
col offload for a7

select sql_id, child_number child, plan_hash_value plan_hash, executions execs,
(elapsed_time/1000000)/decode(nvl(executions,0),0,1,executions)/
decode(px_servers_executions,0,1,px_servers_executions/
decode(nvl(executions,0),0,1,executions)) avg_etime,
px_servers_executions/decode(nvl(executions,0),0,1,executions) avg_px,
decode(IO_CELL_OFFLOAD_ELIGIBLE_BYTES,0,'No','Yes') Offload,
decode(IO_CELL_OFFLOAD_ELIGIBLE_BYTES,0,0,
100*(IO_CELL_OFFLOAD_ELIGIBLE_BYTES-IO_INTERCONNECT_BYTES)
/decode(IO_CELL_OFFLOAD_ELIGIBLE_BYTES,0,1,IO_CELL_OFFLOAD_ELIGIBLE_BYTES)) "IO_SAVED%",
sql_text
from v$sql s
where upper(sql_text) like upper(nvl('&sql_text',sql_text))
and sql_text not like 'BEGIN :sql_text := %'
and sql_text not like '%IO_CELL_OFFLOAD_ELIGIBLE_BYTES%'
and sql_id like nvl('&sql_id',sql_id)
order by 1, 2, 3
/

SYS@SANDBOX1> select avg(pk_col) from kso.skew3 where col1 < 0;
AVG(PK_COL)

```

■ 深入理解 Oracle Exadata

```
-----
1849142.5

Elapsed: 00:00:00.07
SYS@SANDBOX1> alter session set cell_offload_processing=false;

Session altered.

Elapsed: 00:00:00.00
SYS@SANDBOX1> select avg(pk_col) from kso.skew3 where col1 < 0;

AVG(PK_COL)
-----
1849142.5

Elapsed: 00:00:49.68
SYS@SANDBOX1> @fsx4
Enter value for sql_text: select avg(pk_col) from kso.skew3 where col1 < 0
Enter value for sql_id:

SQL_ID          CHILD OFFLOAD IO_SAVED_% AVG_ETIME  SQL_TEXT
-----
a6j7wgqf84jvg    0 Yes      100.00      .07 select avg(pk_col) from kso.skew3
a6j7wgqf84jvg    1 No        .00      49.68 select avg(pk_col) from kso.skew3

Elapsed: 00:00:00.04
```

你可以看到在 `fsx` 脚本中, `OFFLOAD` 列仅仅是做了一个 `DECODE` 去检查 `IO_CELL_OFFLOAD_ELIGIBLE_BYTES` 列的值是不是大于 0, `IO_SAVED_%` 列是从 `IO_INTERCONNECT_BYTES` 列值中计算得来的, 它显示了多少数据会被返回到数据库服务器中。例子中展示了同样的一条语句, 一次执行了智能扫描, 一次没有。我们使用了 `CELL_OFFLOAD_PROCESSING` 参数来打开或者关闭智能扫描选项, 修改该参数将会导致原来的游标无效, 因为优化器环境发生了变化了, 这就产生了该条语句的两个子游标, 在我们的例子中, 脚本输出显示了其中一个版本的语句使用了智能扫描, 只消耗了不到 1 秒钟的时间, 而第二个语句没有使用智能扫描, 花费了将近 1 分钟的时间。

`fsx` 脚本使用到的技术在大多数时候都工作良好, 但是在某些情况下, 一个子游标被反复使用, 可能其中一些执行使用了智能扫描, 而另一些则没有用到。虽然这并不常见, 但是由于 `IO_CELL_OFFLOAD_ELIGIBLE_BYTES` 保存的是所有执行的累计值, 就可能会引起疑惑, 也就是说每次执行都把自己的数值累加到原值上成为一个总值, 当一些执行用到了智能扫描而一些没用到的时候, `IO_CELL_OFFLOAD_ELIGIBLE_BYTES` 字段值就会大于 0, 这是一个相当罕见的现象,

你可能永远不会碰上，不过，下面是一个例子。

```
SYS@SANDBOX1> alter session set "_serial_direct_read"=true;
```

Session altered.

Elapsed: 00:00:00.01

```
SYS@SANDBOX1> -- execution 1
```

```
SYS@SANDBOX1> select avg(pk_col) from kso.skew a where col1 > 0;
```

AVG(PK_COL)

16093748.8

1 row selected.

Elapsed: 00:00:03.51

```
SYS@SANDBOX1> @fsx3
```

Enter value for sql_text: %skew%

Enter value for sql_id:

SQL_ID	CHILD	OFFLOAD	EXECS	ELIGIBLE_BYTES	SQL_TEXT
05cq2hb1r37tr	0	Yes	1	38797312	select avg(pk_col) f

1 row selected.

Elapsed: 00:00:00.01

```
SYS@SANDBOX1> alter session set "_serial_direct_read"=false;
```

Session altered.

Elapsed: 00:00:00.00

```
SYS@SANDBOX1> -- execution 2
```

```
SYS@SANDBOX1> select avg(pk_col) from kso.skew a where col1 > 0;
```

AVG(PK_COL)

16093748.8

1 row selected.

■ 深入理解 Oracle Exadata

Elapsed: 00:00:04.71

SYS@SANDBOX1> @fsx3

Enter value for sql_text: %skew%

Enter value for sql_id:

SQL_ID	CHILD	OFFLOAD	EXECS	ELIGIBLE_BYTES	SQL_TEXT
05cq2hb1r37tr	0	Yes	2	38797312	select avg(pk_col) f

1 row selected.

Elapsed: 00:00:00.01

SYS@SANDBOX1>

SYS@SANDBOX1> alter session set "_serial_direct_read"=true;

Session altered.

Elapsed: 00:00:00.01

SYS@SANDBOX1> -- execution 3

SYS@SANDBOX1> select avg(pk_col) from kso.skew a where col1 > 0;

AVG(PK_COL)

16093748.8

1 row selected.

Elapsed: 00:00:03.54

SYS@SANDBOX1> @fsx3

Enter value for sql_text: %skew%

Enter value for sql_id:

SQL_ID	CHILD	OFFLOAD	EXECS	ELIGIBLE_BYTES	SQL_TEXT
05cq2hb1r37tr	0	Yes	3	58195968	select avg(pk_col) f

1 row selected.

Elapsed: 00:00:00.01

在这个例子中，我们使用了 `_SERIAL_DIRECT_READ` 参数来禁用智能扫描，此参数不会导致游标无效，所以我们的三次执行都使用了相同的游标，如果你仔细看第二次执行，可能会猜到它没有使用智能扫描，因为它比前一次执行要慢，你可以通过查看 `Eligible Byte Count` 并未增加来验证你的猜测。但是，`fsx` 脚本只是简单地检查 `IO_CELL_OFFLOAD_ELIGIBLE_BYTES` 列值是否大于 0，因为两次执行之间该列值保持未变，所以 `fsx` 脚本认为语句使用了智能扫描，实际上第二次执行并未使用。所以请记住这个指标是一个游标所有执行次数的累计值，在上面这个例子中，我们人工设置了单个游标有时使用智能扫描而有时又不使用这样一个不常见的情况，这在真实环境中很少会发生。

`fsx` 脚本演示的技术提供了除跟踪或会话统计值之外的一种非常有用的方法来验证是否确实使用了智能扫描，这个方法最大的好处就是你无须实时地捕获智能扫描事件。`IO_CELL_OFFLOAD_ELIGIBLE_BYTES` 字段存在于 `V$SQL` 视图和一些相关视图中，这意味着也同样会被 AWR 获取，这就提供给了我们关于 SQL 语句如何被执行的历史视图，这是我们主要使用的工具之一，用来快速但是并不严谨地验证是否使用了智能扫描。

2.5.4 SQL 监控

这是另一个非常有用的工具，`REPORT_SQL_MONITOR` 存储过程是 11g 中新增的实时 SQL 监控功能的一部分，它被整合在 `DBMS_SQLTUNE` 包中，提供了大量信息，不仅仅包括一个语句是否被卸载，还包括耻在执行计划的哪一步被卸载。下面是一个例子（其中第一次无卸载操作，然后是同样的语句，第二次有卸载操作）。

```
SYS@SANDBOX1> alter session set cell_offload_processing=false;
```

```
Session altered.
```

```
SYS@SANDBOX1> set echo off
```

```
SYS@SANDBOX1> @avgskew3
```

```
AVG(PK_COL)
```

```
-----
```

```
16093750.2
```

```
SYS@SANDBOX1> @fsx4
```

```
Enter value for sql_text: %skew3%
```

```
Enter value for sql_id:
```

```
SQL_ID          CHILD OFFLOAD IO_SAVED_% AVG_ETIME SQL_TEXT
```

```
-----
```


■ 深入理解 Oracle Exadata

```
6uutdmqr72smc      0 No      .00      57.95 select /* avgskew3.sql */ avg(pk_col) fr
```

```
SYS@SANDBOX1> @report_sql_monitor
```

```
Enter value for sid:
```

```
Enter value for sql_id: 6uutdmqr72smc
```

```
Enter value for sql_exec_id:
```

```
REPORT
```

```
-----
```

```
SQL Monitoring Report
```

```
SQL Text
```

```
-----
```

```
select /* avgskew3.sql */ avg(pk_col) from kso.skew3 a where col1 > 0
```

```
Global Information
```

```
-----
```

```
Status                : DONE (ALL ROWS)
Instance ID           : 1
Session               : SYS (3:5465)
SQL ID                : 6uutdmqr72smc
SQL Execution ID      : 16777216
Execution Started     : 03/15/2011 15:26:11
First Refresh Time    : 03/15/2011 15:26:19
Last Refresh Time     : 03/15/2011 15:27:09
Duration              : 58s
Module/Action         : sqlplus@enkdb01.enkitec.com (TNS V1-V3)/-
Service               : SYS$USERS
Program               : sqlplus@enkdb01.enkitec.com (TNS V1-V3)
Fetch Calls           : 1
```

```
Global Stats
```

```
=====
```

Elapsed	Cpu	IO	Application	Cluster	Fetch	Buffer	Read	Read
Time(s)	Time(s)	waits(s)	waits(s)	waits(s)	Calls	Gets	Reqs	Bytes
58	35	23	0.00	0.00	1	2M	15322	15GB

```
=====
```

```
SQL Plan Monitoring Details (Plan Hash value=2684249835)
```

```
=====
```

Id	Operation	Name	...	Activity	Activity Detail
----	-----------	------	-----	----------	-----------------

```
=====
```

			(%)	(# samples)
0	SELECT STATEMENT			
1	SORT AGGREGATE		41.38	Cpu (24)
2	TABLE ACCESS STORAGE FULL	SKEW3	58.62	Cpu (11)
				direct path read (23)

SYS@SANDBOX1> @ss_on

SYS@SANDBOX1> alter session set cell_offload_processing=true;

Session altered.

SYS@SANDBOX1> @avgskew3

AVG(PK_COL)

16093750.2

SYS@SANDBOX1> @fsx4

Enter value for sql_text: %skew3%

Enter value for sql_id:

SQL_ID	CHILD	OFFLOAD	IO_SAVED_%	AVG_ETIME	SQL_TEXT
6uutdmqr72smc	0	Yes	71.85	34.54	select /* avgskew3.sql */ avg(pk_col) fr

SYS@SANDBOX1> @report_sql_monitor

Enter value for sid:

Enter value for sql_id: 6uutdmqr72smc

Enter value for sql_exec_id:

REPORT

SQL Monitoring Report

SQL Text

select /* avgskew3.sql */ avg(pk_col) from kso.skew3 a where col1 > 0

■ 深入理解 Oracle Exadata

Global Information

```

-----
Status           : DONE (ALL ROWS)
Instance ID      : 1
Session          : SYS (3:5467)
SQL ID           : 6uutdmqr72smc
SQL Execution ID : 16777219
Execution Started : 03/15/2011 15:36:11
First Refresh Time : 03/15/2011 15:36:15
Last Refresh Time : 03/15/2011 15:36:45
Duration         : 34s
Module/Action    : sqlplus@enkdb01.enkitec.com (TNS V1-V3)/-
Service          : SYS$USERS
Program          : sqlplus@enkdb01.enkitec.com (TNS V1-V3)
Fetch Calls      : 1
  
```

Global Stats

```

=====
| Elapsed | Cpu | IO | Application | Fetch | Buffer | Read | Read | Cell |
| Time(s) | Time(s) | waits(s) | Waits(s) | Calls | Gets | Reqs | Bytes | Offload |
=====
| 35 | 31 | 3.85 | 0.00 | 1 | 2M | 18422 | 15GB | 71.83% |
=====
  
```

SQL Plan Monitoring Details (Plan Hash Value=2684249835)

```

=====
| Id | Operation | Name | | Cell | Activity | Activity Detail |
| | | | | Offload | (%) | (# samples) |
=====
| 0 | SELECT STATEMENT | | | | | |
| 1 | SORT AGGREGATE | | | | 50.00 | Cpu (17) |
| 2 | TABLE ACCESS STORAGE FULL | SKEW3 | | 71.83% | 50.00 | Cpu (12) |
| | | | | | | cell smart tab... (5)|
=====
  
```

注意我们从报告中去掉了许多列，因为完整的结果实在太宽了，并不能很漂亮地排版到本书中。你可以看到报告中显示了哪一步是卸载（Cell Offload）、执行时间都耗费在哪些地方（Activity %）、耗费时间去做什么（Activity Detail）。这对于一个在很多步中都适合卸载的复杂语句，会有极大帮助。还要说明的是，对于并行操作语句和那些优化器预计将运行很长时间的语句都会自动监控，如果 Oracle 没有自动监控你需要的语句，你可以使用 MONITOR 提示来告诉 Oracle 去监控。

2.6 参数

与卸载相关有一些参数，最主要的是 `CELL_OFFLOAD_PROCESSING`，用来打开或者关闭卸载，此外还有几个不太重要的。表 2-2 列出了影响卸载的非隐含参数（在 Oracle 数据库版本 11.2.0.2 中）。我们同时也列出了控制着重要功能的隐含参数 `_SERIAL_DIRECT_READ`。

表 2-2 控制卸载的数据库参数

参 数	默 认 值	描 述
<code>cell_offload_compaction</code>	ADAPTIVE	保留为将来使用，联机描述为“存储数据报打包策略”
<code>cell_offload_decryption</code>	TRUE	控制解密操作是否可以被卸载。注意如果此参数设置为 FALSE，那么就会完全禁用在加密数据上的智能扫描
<code>cell_offload_parameters</code>		保留为将来使用
<code>cell_offload_plan_display</code>	AUTO	通过 XPLAN 控制 Exadata 的操作名称是否可以被使用在执行计划输出中。AUTO 表示只有在使用 Exadata 存储时才会显示
<code>cell_offload_processing</code>	TRUE	打开或者关闭卸载
<code>_serial_direct_read</code>	AUTO	控制串行直接路径读取机制。有效值为 AUTO、TRUE、FALSE、ALWAYS 和 NEVER

除了这些普通的 Oracle 批准的参数外，还有不少所谓的隐含参数也会影响卸载，下面的列表列出了所有存储节点的参数，包括了隐含参数以及描述。

```
SYS@POC1> @parmsd
Enter value for parameter: cell
Enter value for isset:
Enter value for show_hidden: Y
```

NAME	DESCRIPTION
-----	-----
<code>cell_offload_compaction</code>	Cell packet compaction strategy
<code>cell_offload_decryption</code>	enable SQL processing offload of encrypted data to cells
<code>cell_offload_parameters</code>	Additional cell offload parameters
<code>cell_offload_plan_display</code>	cell offload explain plan display
<code>cell_offload_processing</code>	enable SQL processing offload to cells
<code>_allow_cell_smart_scan_attr</code>	Allow checking smart_scan_capable Attr
<code>_cell_fast_file_create</code>	Allow optimized file creation path for cells
<code>_cell_fast_file_restore</code>	Allow optimized rman restore for cells
<code>_cell_file_format_chunk_size</code>	cell file format chunk size in MB
<code>_cell_index_scan_enabled</code>	enable CELL processing of index FFS

■ 深入理解 Oracle Exadata

<code>_cell_offload_capabilities_enabled</code>	specifies capability table to load
<code>_cell_offload_hybridcolumnar</code>	Query offloading of hybrid columnar compressed tables to exadata
<code>_cell_offload_predicate_reordering_enabled</code>	enable out-of-order SQL processing offload to cells
<code>_cell_offload_timezone</code>	enable timezone related SQL processing offload to cells
<code>_cell_offload_virtual_columns</code>	enable offload of predicates on virtual columns to cells
<code>_cell_range_scan_enabled</code>	enable CELL processing of index range scans
<code>_cell_storidx_mode</code>	Cell Storage Index mode
<code>_db_check_cell_hints</code>	
<code>_disable_cell_optimized_backups</code>	disable cell optimized backups
<code>_kcfis_cell_passthru_enabled</code>	Do not perform smart IO filtering on the cell
<code>_kcfis_kept_in_cellfc_enabled</code>	Enable usage of cellsrv flash cache for kept objects
<code>_kcfis_nonkept_in_cellfc_enabled</code>	Enable use of cellsrv flash cache for non-kept objects

22 rows selected.

我们在本章的例子中已经使用过其中的几个参数，在后续章节中还会使用一些。另外，还有一些 kcfis（kernel file intelligent storage）参数，下面是这些参数的列表。

```
SYS@SMITHERS> @parmsd
Enter value for parameter: kcfis
Enter value for isset:
Enter value for show_hidden: Y
```

NAME	DESCRIPTION
<code>_kcfis_block_dump_level</code>	Smart IO block dump level
<code>_kcfis_caching_enabled</code>	enable kcfis intra-scan session caching
<code>_kcfis_cell_passthru_enabled</code>	Do not perform smart IO filtering on the cell
<code>_kcfis_control1</code>	Kcfis control1
<code>_kcfis_control2</code>	Kcfis control2
<code>_kcfis_control3</code>	Kcfis control3

<code>_kcfis_control4</code>	Kcfis control4
<code>_kcfis_control5</code>	Kcfis control5
<code>_kcfis_control6</code>	Kcfis control6
<code>_kcfis_disable_platform_decryption</code>	Don't use platform-specific decryption on the storage cell
<code>_kcfis_dump_corrupt_block</code>	Dump any corrupt blocks found during smart IO
<code>_kcfis_fast_response_enabled</code>	Enable smart scan optimization for fast response (first rows)
<code>_kcfis_fast_response_initiosize</code>	Fast response - The size of the first IO in logical blocks
<code>_kcfis_fast_response_iosizemult</code>	Fast response - (next IO size = current IO size * this parameter)
<code>_kcfis_fast_response_threshold</code>	Fast response - the number of IOs after which smartIO is used
<code>_kcfis_fault_control</code>	Fault Injection Control
<code>_kcfis_io_prefetch_size</code>	Smart IO prefetch size for a cell
<code>_kcfis_ioreqs_throttle_enabled</code>	Enable Smart IO requests throttling
<code>_kcfis_kept_in_cellfc_enabled</code>	Enable usage of cellsrv flash cache for kept objects
<code>_kcfis_large_payload_enabled</code>	enable large payload to be passed to cellsrv
<code>_kcfis_max_cached_sessions</code>	Sets the maximum number of kcfis sessions cached
<code>_kcfis_max_out_translations</code>	Sets the maximum number of outstanding translations in kcfis
<code>_kcfis_nonkept_in_cellfc_enabled</code>	Enable use of cellsrv flash cache for non-kept objects
<code>_kcfis_oss_io_size</code>	KCFIS OSS I/O size
<code>_kcfis_rdbms_blockio_enable</code>	Use block IO instead of smart IO in the smart IO module on RDBMS
<code>_kcfis_read_buffer_limit</code>	KCFIS Read Buffer (per session) memory limit in bytes
<code>_kcfis_spawn_debugger</code>	Decides whether to spawn the debugger at kcfis

■ 深入理解 Oracle Exadata

	initialize
_kcfis_stats_level	sets kcfis stats level
_kcfis_storageidx_diag_mode	Debug mode for storage index on the cell
_kcfis_storageidx_disabled	Don't use storage index optimization on the storage cell
_kcfis_test_control1	kcfis tst control1
_kcfis_trace_bucket_size	KCFIS tracing bucket size in bytes
_kcfis_trace_level sets	kcfis tracing level
_kcfis_work_set_appliances	Working Set of appliances in a KCFIS session

34 rows selected.

注意，如果没有事先跟 Oracle 技术支持人员讨论过，那么隐含参数不应该在产品环境中使用，不过这些参数确实提供了有意义的线索，让我们知道某些 Exadata 功能是如何工作、如何控制的。

2.7 总结

卸载绝对是 Exadata 的独门兵器，Exadata 的硬件架构做了良好的工作，在存储层的提供数据能力和数据库层的消费数据能力之间提供了很好的平衡，与此同时，Exadata 的软件带来了更大的性能提升，智能扫描正是这些收益的主要原因，大多数优化的首要重点是减少存储层和数据库层之间传输的数据量。