

**研究生学位论文开题报告**

**报告题目 内存数据网格Cache-Aside模式的设计与实现**

**学生姓名 江勇 学号 2013E8015061092**

**指导教师 徐罡 职称 副研究员**

**学位类别 工程硕士**

**学科专业 　 软件工程**

**研究方向 　 分布式软件系统工程**

**培养单位 中国科学院软件研究所**

**填表日期 2015年3月23日**

**中国科学院大学制**

**填 表 说 明**

1. 本表内容须真实、完整、准确。
2. “学位类别”名称填写：哲学博士、教育学博士、理学博士、工学博士、农学博士、医学博士、管理学博士，哲学硕士、经济学硕士、法学硕士、教育学硕士、文学硕士、理学硕士、工学硕士、农学硕士、医学硕士、管理学硕士等。
3. “学科专业”名称填写： “二级学科”全称。

# 选题的背景及意义

## 1.1 关系型数据库的局限性

随着网络技术的发展，用户数量和网络流量出现了爆炸性的增长，网络化应用面临超大规模的用户量和数据量[1]， 全球每天通过Internet网络传输的电子邮件多达2100亿封；Facebook每月新增10亿照片和1000万个视频；腾讯公司注册用户超过7亿，同时在线人数超过1亿[2]。

如此大流量的网络访问造成的大规模高并发的数据读写给数据存储系统带来挑战，基于关系型数据库系统扩展技术面临着诸多难题，主要包括：

（1）存储灵活性不足。关系型数据库按照严格的关系范式进行建模，能够有效避免数据的冗余存储。但却无法自由更改表结构和列存储类型，在以非结构化数据占主导地位的大数据时代，这严重限制了关系型数据库的可应用场景。

（2）扩展能力不足。关系型数据库的扩展方式一般有分库分表、主从复制、异构复制等，但这些技术存在业务逻辑紧耦合、单点瓶颈、放松数据一致性等缺点，导致系统扩展能力受到限制。

（3）高并发写能力不足。目前，关系型数据库主要以硬盘作为存储介质，面向关系型数据库的集群读写分离设计、分布式缓存等系统扩展技术仅适用于读密集型操作，当面临高并发写请求时，磁盘写入瓶颈会严重增加系统响应延迟。

图1.1展示了一个典型的Web应用的系统架构。整个系统包含了客户端、负载均衡器、应用服务器以及数据库服务器等主要组成部分。从图中可以看出，数据库服务器不能像应用服务器那样采用集群的方式来应对大负载，极易成为整个系统的性能瓶颈。有一种策略是将数据以冗余备份的方式存放在若干服务器中构成一个服务器集群。这种方案可以在一定程度上缓解数据库服务器的读压力，可是当系统中有频繁的数据更新操作时，由于要保障数据一致性，整个系统会表现出更加糟糕的性能。

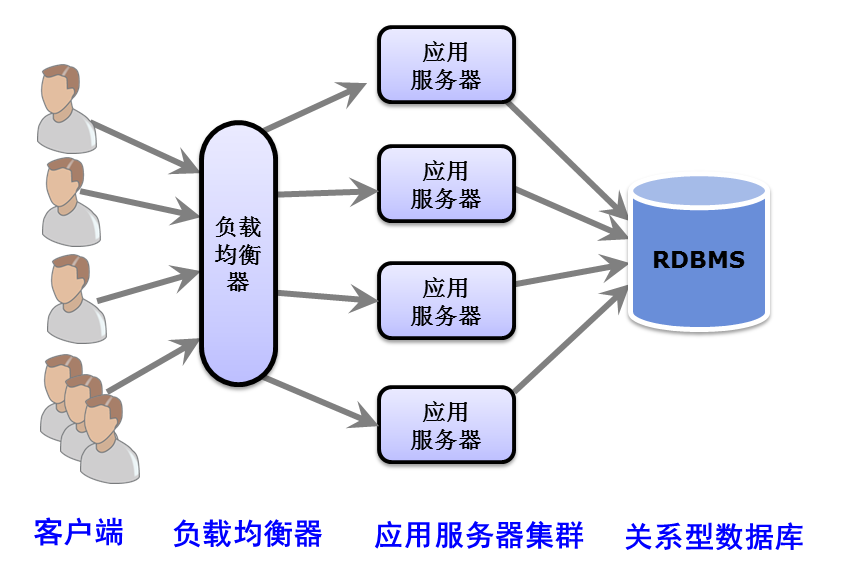


图1.1传统Web应用系统架构

与传统的关系型数据库相对应的是将数据直接存放在内存中进行操作的内存数据库[3][4][5]。内存数据库具有读取速度快、性能高等优点，也可以在一定程度上缓解数据库系统的瓶颈问题。然而，由于内存数据库的数据模型跟传统的关系型数据库是一样的，它也没有克服传统关系型数据库具有的缺乏灵活性和拓展性的缺点。在这种情况下，靠增加单个服务器的计算能力、存储能力和网络带宽等方法是难以应对不断增加的并发用户的规模。在实践中，内存数据库并没有被广泛应用。

## 1.2 内存数据网格的发展与不足

图灵奖获得者Jim Gray认为，应该将随机操作交给内存，只给硬盘留下顺序操作[6]。在上述背景下，工业界提出内存数据网格(In Memory Data Grid, 简称IMDG)的新型的数据处理引擎，它将应用的持久化数据存储在非关系型(主要是key/value)的分布式内存中。与传统关系型数据库相比，内存数据网格具有如下优势：

（1）灵活的数据模型。内存数据网格采用键值对的存储模型，无需提前为欲存储数据建立固定的数据结构，可以灵活的自定义数据存储格式。该特性可以很好的满足非结构化数据的存储需求，更适用于对海量数据的处理。

（2）易扩展。内存数据网格基于非关系型数据模型设计，去除了数据之间的关联关系，使数据库集群架构的设计更为灵活，可以在网格集群中动态加入和删除服务节点，同时由于数据之间没有关联，内存数据网格可以将系统负载更均匀的分配到不同结点，而避免产生单点瓶颈。

（3）内存存储。内存数据网格基于内存进行数据存储，避免了磁盘读写速度的短板，可以更好地应对海量数据、高并发情况下的数据访问请求。

正是由于内存数据网格具有这些优点，在内存容量不断扩大，价格不断下降的背景下，内存数据网格受到了越来越广泛的关注。虽然如此，但仍然存在一些问题在阻碍内存数据网格的进一步发展，主要有：

（1）单结点存储能力不足。内存数据网格将数据存放在内存之中的一个明显限制是单结点内存无法像磁盘一样提供大容量的存储能力，只能通过网格扩展来弥补该不足，但这无疑提高了保证数据一致性的难度，同时也增加了分布式通信的开销。

（2）对遗留应用兼容能力不足。目前主流的内存数据网格产品大多缺乏对遗留系统兼容性的考虑，对传统的基于结构化查询语言的支持还很不完善，这在很大程度上限制了遗留系统向内存数据网格的迁移，抬高了内存数据网格应用的门槛，不利于内存数据网格的健康发展。

（3）大内存堆GC效率低。JVM会有一个或多个专门的垃圾回收线程，由它们负责清理回收垃圾内存，在早期的GC工作期间，所有其它的线程都被暂停，以免影响垃圾回收工作，严重影响应用性能，虽然经过了改进，如采用了分代技术，但是线程的暂时性休克依然存在。垃圾回收时需要遍历所有可达对象，当堆内存分配较多时，开销不可忽视。遍历完对象引用后，对垃圾的清理和回收也有较大的开销，这部分开销可能包括内存复制块，更新对象引用等等。在使用大内存堆时，GC效率低下也一直限制着内存数据网格性能的进一步提升。

## 1.3 系统设计目标

我国铁路客户服务中心网站12306在二期建设时，利用Cache-Aside模式，实现基于内存的余票和订单的查询，有效缓解了服务器压力，成功让上亿中国公民网上购票。图1.2是12306网上订票系统架构。

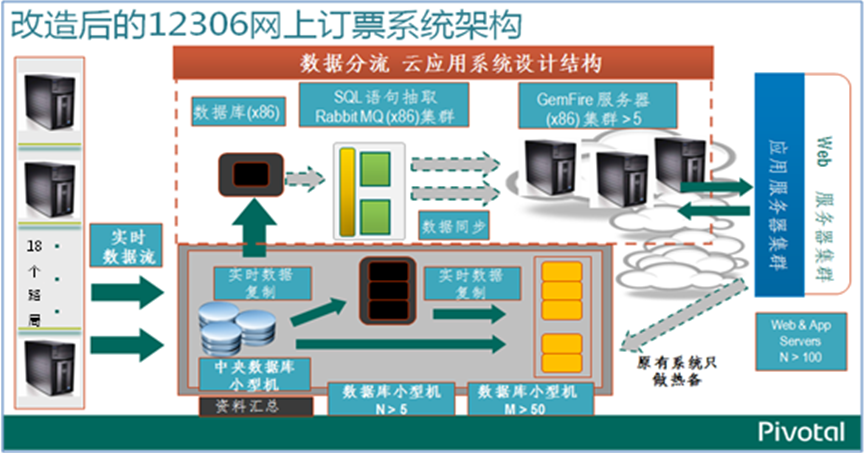


图1.2 改造后的12306网上订票系统架构

考虑到内存数据网格的不足以及12306的成功案例，我们提出内存数据网格的Cache-Aside模式。在Cache-Aside模式下，我们仅加载与系统关键业务逻辑相关的数据，具体数据可由用户配置。在数据访问时，对于未加载数据，直接交由数据库服务器处理，对于已加载的数据，可交由内存数据网格来执行，Cache-Aside只需做到关键业务逻辑的兼容，便可使整个内存数据网格以很低的代价的迁移到现有系统中。如果在Cache-Aside模式下，堆内存依然较大，我们可以采用堆外存储技术来避免垃圾回收机制对大容量堆内存操作带来的时间开销。

# 2 国内外本学科领域的发展现状与趋势

目前工业界和开源社区已有许多内存数据网格产品，其中较为著名的有WebSphere eXtreme Scale[7]、Oracle Coherence[8]、JBoss Infinispan[9]、Terracotta[10]、GigaSpaces[11]和Hazelcast[12]等，JAVA社区也在积极研制内存数据网格标准JSR-347[13]，规范对内存数据网格访问、存储和管理数据的API，并描述其副本、分布式和事务等特性，下面对已有的内存数据网格产品做简要介绍。

## 2.1 Oracle Coherence

Oracle Coherence是Oracle云应用程序基础的关键组件，支持实时的数据分析、内存网格计算以及并行事务和事件处理，支持以线性、动态的方式扩展应用程序。

在数据加载方面，Oracle Coherence 支持以分布式加载的方式进行数据并发加载，以提高吞吐量和CPU 使用率，用户可以觉得那些节点作为对象进行加载，以及数据如何划分给这些节点。

在对遗留应用兼容能力方面，Oracle Coherence参照传统JPA接口实现了Coherence Query Language(CohQL)，CohQL类似于标准JPA接口的一个子集，仅包含轻量级的简单查询语法，对多表的JOIN查询、排序等复杂操作目前暂不支持。

在堆外存储方面，Oracle Coherence将备份数据存储在堆外，这样可以降低垃圾回收时的时间开销，同时因为备份数据的访问频率较低，将数据放于堆外带来的反序列化开销代价很小。

Coherence支持Cache模式和数据网格模式两种使用方式。在Cache模式下，除基于主键的所有请求会直接交给关系数据库处理，Coherence对缓存数据支持Read/Write Through/Behind多种数据读写方式；而数据网格模式下，Coherence通过Filter功能实现请求应答。

## 2.2 GigaSpaces XAP

GigaSpaces XAP是一种在分布式环境中可自动扩展的内存数据网格产品。

在数据加载方面，GigaSpaces 允许用户在客户端可以使用前预加载持久化源中的数据到内存数据网格中。默认情况下GigaSpaces 通过过滤的方式过滤掉不相干数据后，将各个相关数据从数据源并发加载到相关节点。

在对遗留应用兼容能力方面，Gigaspaces同时支持JPA与SQL两种数据访问方式，实现了标准的JDBC接口，目前支持的语法包括AND/OR、逻辑关系运算符、LIKE、ORDER BY等。但是，Gigaspaces对语法的支持并不完善，对于VIEW、INDEX、JOIN、UNION、CASCADE、PRIMARY KEY、NULL、VALUES等常用功能及关键字目前尚未支持，JDBC接口功能存在严重的局限性。

## 2.3 Terracotta

Terracotta是一个基于JVM的内存数据网格解决方案。它提供了包括POJO存储、Http Session复制在内的多种功能。Terracotta通过细粒度对象管理机制来实现并发，使用代码注入（Binary code injection）方法捕获集群节点发出的数据修改请求。它可以增量传输对象中变化的属性，进而能够有效降低网络通信开销。

在对遗留应用兼容能力方面，Terracotta允许用户在不改变遗留系统代码的情况下实现单机Java应用向集群应用的无缝迁移。这可以提升系统的性能，降低应用开发的难度。

在堆外存储技术方面，Terracotta实现了自己的内存管理器从而完全绕过了JVM的垃圾回收，通过BigMemory[14]机制来支持进程内堆外存储。BigMemory在内存存储策略上有别于传统的缓存解决方案。它不把数据存储在Java堆里，从而避免了JVM的GC问题。BigMemory机制使得Terracotta很好地支持大容量的堆内存，避免了垃圾回收机制带来的性能波动。

## 2.4 Hazelcast

Hazelcast是一个开源的内存数据网格解决方案。它支持节点的动态变更和数据的自动重新分配。当一个服务器节点新加入或者离开的时候，数据会重新分配达到新的平衡。

在对遗留应用兼容能力方面，Hazelcast对SQL语法仅仅支持简单查询操作，而对复杂的JOIN、排序、聚集函数和嵌套查询都还没有做到很好的支持。

在堆外存储方面，它支持堆外存储的弹性内存机制，能够克服单个服务器节点对可用内存大小的限制。

## 2.5 EasyCache2.0

EasyCache2.0是由中科院软件研究所软件工程中心自主研发的内存数据网格方案。在数据加载方面，该方案支持数据的全局加载，在对遗留应用兼容能力方面，EasyCache2.0对SQL语法仅支持简单的增删改查和简单的Join查询，对于复杂的排序，聚集函数和嵌套查询都没有做到很好的支持，在JDBC接口方面还有很多不完善的地方。对堆外存储尚不支持。

## 2.6 内存数据网格现状总结

由调研结果可知，内存数据网格都已成为研究的热点并具有巨大产业价值，在大数据时代内存数据网格必然扮演着不可替代的角色，但同时也应看到虽然我们已在关系型数据库上具备了深厚的技术积累，同时在数据切分、分布式存储、SQL翻译等研究领域取得一定的进展，但仍无法满足内存数据网格的技术需求，内存数据网格在系统架构、结构化查询、堆外存储等领域的研究仍面临着巨大的挑战。

# 3 课题主要研究内容、预期目标

内存数据网格系统的主要构成如图3.1所示。本论文主要在EasyCache2.0的基础上设计内存数据网格系统的Cache-Aside架构以及在Cache-Aside架构下对数据的局部自动加载、数据访问的透明集成技术、堆外存储技术等多方面进行研究。

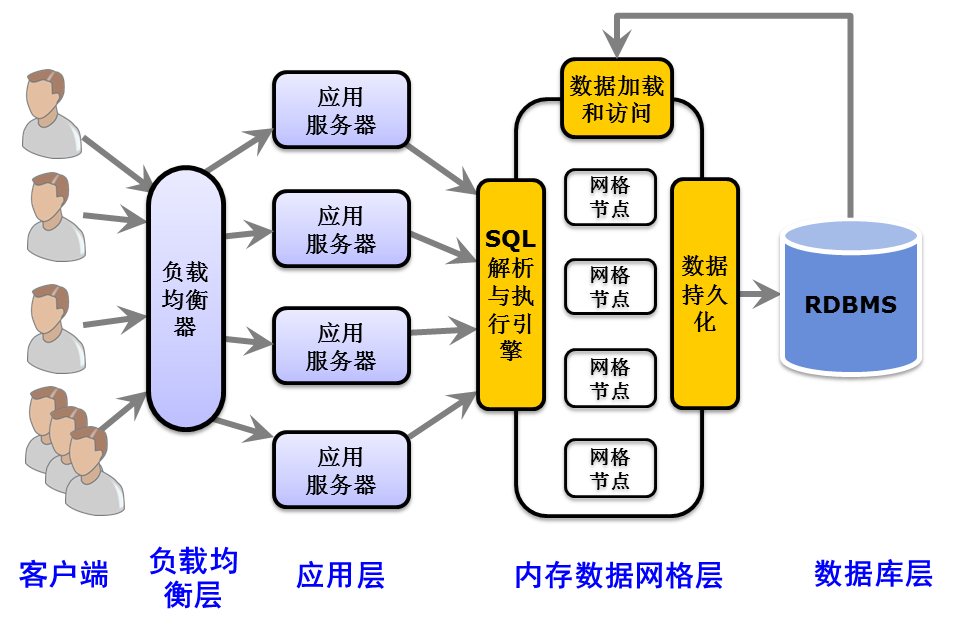


图3.1内存数据网格系统构成

## 3.1 研究内容

### 3.1.1 内存数据网格Cache-Aside系统架构

传统的应用程序中经常使用cache来优化那种需要频繁、重复地访问存储中数据的情景。但一般情况下很难使cache中的数据与底层存储中的数据达到完全一致性，而且由于其依赖关系型数据库，其扩展性也受到了限制。我们首先要做的就是设计一种内存数据网格的Cache-Aside模式，对于关键业务我们都通过网格来处理，而其它业务则直接交由数据库。图3.2是传统架构和内存数据网格架构的对比。

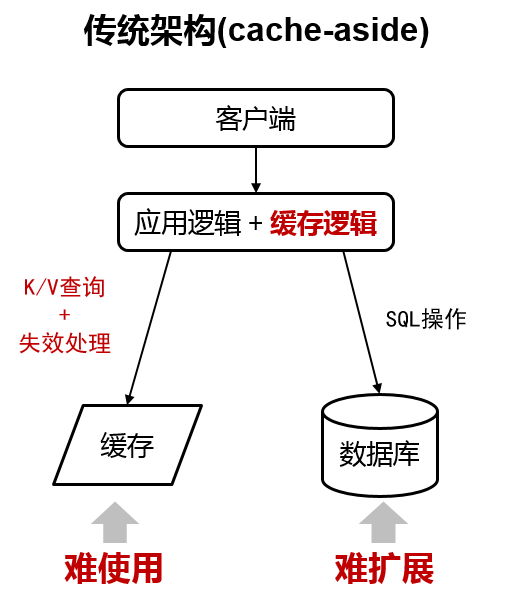
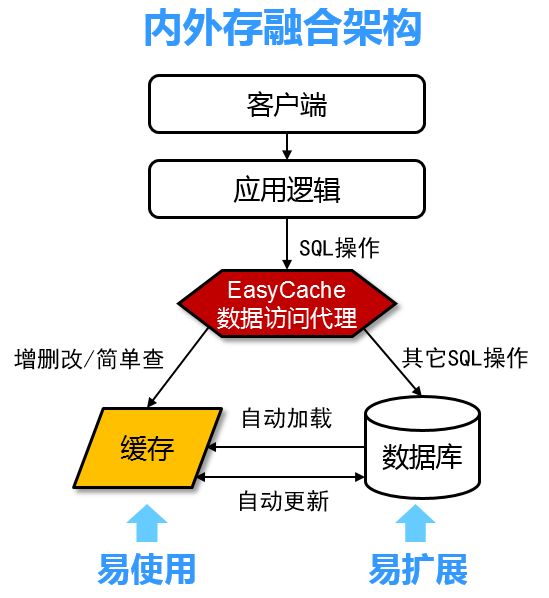


图3.2 传统架构和内存数据网格架构的对比

### 3.1.2 数据局部自动加载

内存数据网格不能持久化的存储数据，系统启动的时候需要将数据从关系型数据库加载到内存数据网格中。然而，两个原因使得这一工作变得比较困难。首先，内存数据网格中的数据遵循key/value 数据模型，而后台关系型数据库中的数据时按照关系型数据模型存储的。在数据加载的过程中，需要解决关系型数据模型到key/value 数据模型的映射问题，即关系型数据库表中的一条记录如何存储到内存数据网格中。其次，内存数据网格产品存储的value 多是将JavaBeans 对象序列化之后的字节流。数据加载的过程需要解决根据数据表的元数据自动构造Java 类和自动生成Java 对象的问题。

在Cache-Aside模式下，并不会加载所有数据，仅加载与业务逻辑密切相关的数据，在这种应用场景下，对分布式缓存各节点之间，数据库与缓存之间的数据通信和同步更新又有了更加严格的要求。

### 3.1.3 数据访问的透明集成技术

除GigaSpaces和EasyCache2.0之外，目前尚没有其他内存数据网格产品对结构化查询语言访问驱动做支持，而GigaSpaces实现的JDBC驱动对索引、更新、排序及联接操作又存在诸多限制，EasyCache2.0对SQL语法和JDBC接口的支持也不够完善，无法避免对遗留应用的源码修改，遗留应用向内存数据网格的迁移代价较大，因此迫切需要内存数据网格探索出一种方案能更好的兼容结构化查询语言。本论文一个主要的研究内容是探索如何实现内存数据网格对结构化查询语言的兼容，实现遗留应用向内存数据网格的无缝迁移。

### 3.1.4 堆外存储优化技术

内存数据网格单点资源管理通过将运用堆外存储技术，通过将数据存储在堆外内存中，避免回收机制带来的性能抖动。但是，堆外技术的缺陷是会带来更多的数据序列化开销，对堆外数据的访问延迟会增加，并且需要系统能够高效的实现对堆外内存的管理，否则对堆外技术的运用会得不偿失。因此，堆外技术的运用应该与数据访问特性相结合。本论文一个主要的研究内容是针对堆外技术的运用，分析探讨什么样的数据适合放置在堆外，是否借鉴Coherence的技术方案，将备份数据存储在堆外，还是通过研究堆外存储技术提高数据的访问性能，采用Terracotta的方式重新设计一种新的内存资源管理器。

## 3.2 预期目标

本论文的预期目标是在EasyCache2.0基础上设计实现一个完整的内存数据网格Cache-Aside产品，针对不同的业务场景，合理运用不同的数据加载、数据访问、数据存储等技术平衡的利用内存、CPU等系统资源，尽最大可能的提升系统整体性能，并能在一定程度上实现对结构化查询语言的兼容，实现对传统应用向内存数据网格系统迁移的技术支持。

# 4 拟采用的研究方法、技术路线、实验方案及其可行性

## 4.1内存数据网格Cache-Aside系统架构

通过对各数据网格产品的系统架构以及Cache-Aside模式的调研，结合内存数据网格扩展性强、内存存储和Cache-Aside模式缓存机制的特点，分析实际的业务需求，在中科院软件所软件工程中心开发设计的EasyCache2.0的基础上，设计一个相对合理的Cache-Aside架构，从后续的几个方面来实现内存数据网格Cache-Aside模式的完整产品。

## 4.2 数据局部自动加载

首先，研究后台关系型数据库的关系型数据模型和内存数据网格中的数据遵循key/value 数据模型，从关系型数据库数据的元数据入手，对于表中具体数据信息，利用代码生成工具CGLib 动态的构造一个类，类的每一个属性跟表的属性对应，属性的名称和类型也一一对应。从而完成关系型数据库数据向内存数据网格数据的转换。

其次，我们可通过配置、过滤等手段动态来决定需要加载的表数据，从而实现数据的局部自动加载。

## 4.3 数据访问的透明集成技术

通过兼容标准JDBC接口来实现对结构化查询语言的支持，对简单的结构化查询语句借鉴Gigaspaces 及Hazelcast对结构化查询语言的兼容技术，在已有技术基础上，进一步实现对ORDER BY，LIMIT等复杂功能的支持；对于其他无法支持或在内存数据网格中支持代价过高的结构化查询请求，将被直接转交给底层关系型数据库进行处理。使用关系型数据库做辅助处理可以极大程度地完善对结构化查询语言的兼容，做到遗留系统向内存数据网格的无缝迁移。

## 4.4 堆外存储优化技术

参照Terracotta的BigMemory，不把数据存储在堆内，而是使用了JDK 1.4引入的DirectByteBuffers，给应用配置很小的堆空间来避免GC问题。并结合采用C4(Continuously Concurrent Compacting Collector)垃圾回收算法[15]将释放内存的过程从应用程序行为和内存分配速率中分离出来，并加以区分。这样就实现了并发运行，即应用程序可以持续运行，而不必等待垃圾回收的完成。

# 5 已有科研基础与所需的科研条件

阅读了大量内存数据网格、数据访问透明集成、数据存储模型的相关文献。

中国科学院软件研究所软件工程技术中心已经开发和实现了内存数据网格EasyCache2.0版本，可以为本论文的研究提供良好的基础平台。

实验室提供服务器集群环境、高性能刀片机，虚拟机，为实验提供充足硬件条件。

# 6 研究工作计划与进度安排

2015.04至2013.05调研内存数据网格、数据访问透明集成、数据存储模型相关文献，设计出内存数据网格Cache-Aside架构。

2015.06至2015.07 研究后台关系型数据库的关系型数据模型和内存数据网格中的key/value 数据模型，以及开源工具cglib， 实现数据的局部自动加载。

2015.08至2015.11研究结构化查询语言的语法语义执行流程及向非关系型数据模型迁移的中限制，在已有系统上增加对ORDER BY 和LIMIT等复杂查询功能的支持。

2015.12至2016.03研究不同存储模型下的堆外存储性能，在系统中添加对堆外存储的支持。

2016.04至2016.05总结系统开发工作，撰写硕士论文。

# 7 参考文献

[1]. Mike Gualtieri and John R.Rymer, The Forrester Wave: Elastic Caching Platforms, Q2. 2010.

[2]. 《中国科学报》 (2013-05-10 第9版 探索周刊)

[3]. Garcia-Molina H, Salem K. Main memory database systems: An overview [J]. Knowledge and Data Engineering, IEEE Transactions on, 1992, 4(6): 509-516.

[4]. DeWitt D J, Katz R H, Olken F, et al. Implementation techniques for main memory database systems[M]. ACM, 1984.

[5]. Lehman T J, Carey M J. Query processing in main memory database management systems [M]. ACM, 1986.

[6]. Gray Jim．Tape is dead disk is tape flash is disk RAM locality is king, Pacific Grove: Microsoft, Gong Show Presentation at Third Biennial, Conference on Innovative Data Systems Research: 1, 2007．

[7]. http://www-01.ibm.com/software/webservers/appserv/extreme scale/

[8]. http://www.oracle.com/technetwork/middleware/coherence/overview/ index.html.

[9]. https://community.jboss.org/en/infinispan

[10]. http://terracotta.org/

[11]. http://www.gigaspaces.com/

[12]. http://www.hazelcast.com/.

[13].JSR 347, Data Grids for the JavaTM Platform. http://jcp.org/en/jsr/detail?id=347.

[14].http://terracotta.org/products/bigmemory

[15].Gil Tene, Balaji Iyengar, Michael Wolf, C4: The Continuously Concurrent Compacting Collector, pages 79-88, NY, New York, USA, 2011, ACM