PaperTime检测报告简明打印版

相似度：23.4%
编　号：NVBCFDIGXJWTSEYG
标　题：作业调度平台中心控制系统的设计与实现
作　者：李佰波
长　度：45990字符
时　间：2019-05-22 21:30:06
比对库：本地库（学术期刊、学位论文、会议论文）；PaperTime云论文库；互联网

本地库相似资源（学术期刊、学位论文、会议论文）
1. 相似度：1.26%　篇名：《基于MongoDB的海量空间数据存储和并行》
来源：《地理空间信息》　年份：2014　作者：张恩
2. 相似度：0.78%　篇名：《内嵌型PPT下载妙招》
来源：《网络运维与管理》　年份：2013　作者：牟晓东
3. 相似度：0.24%　篇名：《基于服务器推送技术的在线交流平台的研究与实现》
来源：《燕山大学硕士学位论文》　年份：2015　作者：梁浩
4. 相似度：0.22%　篇名：《Java异常处理》
来源：《电脑编程技巧与维护》　年份：2012　作者：吴翠鸿
5. 相似度：0.17%　篇名：《基于MongoDB的EHR存储方案研究与设计》
来源：《中国数字医学》　年份：2013　作者：刘愉
6. 相似度：0.15%　篇名：《一种实时集群系统负载均衡通用模型的研究及应用》
来源：《湖南大学硕士学位论文》　年份：2008　作者：王俊峰
7. 相似度：0.13%　篇名：《教师个人工作总结》
来源：《新课程学习：上》　年份：2013　作者：安艳
8. 相似度：0.12%　篇名：《基于Agent的三次采油动态集成平台的研究与实现》
来源：《东北石油大学硕士学位论文》　年份：2016　作者：李少龙
9. 相似度：0.08%　篇名：《流媒体服务器集群负载均衡策略的研究》
来源：《西安邮电大学硕士学位论文》　年份：2017　作者：王钊
10. 相似度：0.08%　篇名：《基于龙芯的嵌入式智能家居系统的研究》
来源：《中国石油大学(华东)硕士学位论文》　年份：2010　作者：李永
11. 相似度：0.08%　篇名：《并行查询加速器的设计与实现》
来源：《长春理工大学硕士学位论文》　年份：2010　作者：张俊毅
12. 相似度：0.08%　篇名：《基于MVC模式的Web前端框架关键技术研究与实现》
来源：《中国海洋大学硕士学位论文》　年份：2014　作者：乔淑夷
13. 相似度：0.07%　篇名：《基于HTTP长连接的消息推送平台的研究与实现》
来源：《华中科技大学硕士学位论文》　年份：2012　作者：杨文婷
14. 相似度：0.07%　篇名：《《算法描述与设计》教学设计》
来源：《中国信息技术教育》　年份：2014　作者：卜囡
15. 相似度：0.06%　篇名：《云数据中心环境下并行应用与并行负载调度算法研究》
来源：《国防科学技术大学博士学位论文》　年份：2014　作者：邓科峰
16. 相似度：0.05%　篇名：《基于Quartz的网管系统任务调度的实现》
来源：《电脑开发与应用》　年份：2011　作者：王崟
17. 相似度：0.04%　篇名：《基于TCP重发与加权响应时间算法的负载均衡器的研究与实现》
来源：《中南大学硕士学位论文》　年份：2012　作者：李金
18. 相似度：0.04%　篇名：《虚拟机服务器技术在网站建设中的应用》
来源：《复旦大学硕士学位论文》　年份：2012　作者：陈昌锋
19. 相似度：0.04%　篇名：《基于Cougaar的智能Agent研究与实现》
来源：《哈尔滨工业大学硕士学位论文》　年份：2006　作者：吴迪
20. 相似度：0.03%　篇名：《流程工业实时数据库分布式服务框架的设计与开发》
来源：《浙江大学硕士学位论文》　年份：2015　作者：李龙
21. 相似度：0.03%　篇名：《基于Linux的高可用集群管理与监控系统设计与实现》

来源：《哈尔滨工业大学硕士学位论文》 年份：2014 作者：丑玉锋
22. 相似度：0.03% 篇名：《支持集群的消息中间件的研究与设计》
来源：《中国地质大学（北京）硕士学位论文》 年份：2009 作者：秦攀科
23. 相似度：0.03% 篇名：《基于分布式测控系统的服务器集群负载均衡技术的研究》
来源：《南京航空航天大学硕士学位论文》 年份：2006 作者：吴欢欢
24. 相似度：0.03% 篇名：《基于DFR的多Agent社会行为机制及应用研究》
来源：《苏州大学硕士学位论文》 年份：2010 作者：丁家振
25. 相似度：0.03% 篇名：《Java异常处理机制探讨》
来源：《科技视界》 年份：2013 作者：杨毅
26. 相似度：0.03% 篇名：《基于IVCE的数据分发网络的研究与实现》
来源：《北京邮电大学硕士学位论文》 年份：2015 作者：张哲宇
27. 相似度：0.03% 篇名：《基于LVS数据库集群负载均衡算法的研究》
来源：《曲阜师范大学硕士学位论文》 年份：2017 作者：王超

PaperTime云论文库(知网，万方，维普，百度文库等镜像）

1. 相似度：0.66% 标题：《大学生毕业感言范文三篇_百度文库》
来源：http://wenku.baidu.com/view/65365dfbb8d528ea81c758f5f61fb7360b4c2bac.html
2. 相似度：0.46% 标题：《服务器集群技术.ppt》
来源：https://www.taodocs.com/p-75972935.html
3. 相似度：0.28% 标题：《Flight mode recognition method of the unmanned aerial vehicle based...》
来源：http://en.cnki.com.cn/Article_en/CJFDTotal-YQXB201609011.htm
4. 相似度：0.16% 标题：《一种支持多任务高效处理的遥感产品生产线架构研究-维普期刊 中文...》
来源：http://lib.cqvip.com/Qikan/Article/Detail?id=1003452233&from=Qikan_Search_Index
5. 相似度：0.15% 标题：《超算CAE云平台作业调度管理系统的研究与实现 - 道客巴巴》
来源：http://www.doc88.com/p-0873595454796.html
6. 相似度：0.13% 标题：《德育答辩范文 - 道客巴巴》
来源：http://www.doc88.com/p-799540964545.html
7. 相似度：0.10% 标题：《Bottleneck and breakthrough of higher vocational ...》
来源：http://en.cnki.com.cn/Article_en/CJFDTotal-MXZD201702003.htm
8. 相似度：0.06% 标题：《...recognition method of the unmanned aerial vehicle based telemetric》
来源：http://d.old.wanfangdata.com.cn/Periodical/yqyb201609011
9. 相似度：0.04% 标题：《集群监控中的检测技术对策 - 道客巴巴》
来源：http://www.doc88.com/p-2486643607316.html
10. 相似度：0.04% 标题：《多集群作业调度系统MCJSS的设计与实现 - 豆丁网》
来源：https://www.docin.com/p-1330589414.html

互联网相似资源(博客，百科，论坛，新闻等）

1. 相似度：6.65% 标题：《数据平台作业调度系统详解 理论篇| 实践,德仔网》
来源：http://tech.dezai.cn/Detail.aspx?AI=96393&CI=4
2. 相似度：1.44% 标题：《数据平台调度系统- 简书》
来源：https://www.jianshu.com/p/d5606376ad88
3. 相似度：1.39% 标题：《单机系统与集群系统 - 简书》
来源：https://www.jianshu.com/p/231fdee35e6a
4. 相似度：1.02% 标题：《使用JAX-RS创建RESTful Web Service - 搜索技术 - 博客园》
来源：https://www.cnblogs.com/ydxblog/p/7993991.html
5. 相似度：0.79% 标题：《毕业感言_百度知道》
来源：https://zhidao.baidu.com/question/452193352435677045.html
6. 相似度：0.65% 标题：《阻塞队列 LinkedBlockingQueue Ruthless》
来源：https://www.cnblogs.com/linjiqin/p/5128048.html
7. 相似度：0.58% 标题：《java UncaughtExceptionHandler 处理线程意外中止- freeliver54 - 博客园》
来源：http://www.cnblogs.com/freeliver54/archive/2011/10/17/2215423.html
8. 相似度：0.57% 标题：《关于集群监控| 李乾坤的博客》

来源：http://qiankunli.github.io/2014/09/10/monitor.html
9. 相似度：0.45% 标题：《青年英才耀龙江:哈工程赵彬彬破解海浪奥秘为船舶安全加码_研究》
来源：http://www.sohu.com/a/247263650_172952
10. 相似度：0.43% 标题：《dblp: ISPA 2005》
来源：https://dblp.uni-trier.de/db/conf/ispa/ispa2005.html
11. 相似度：0.43% 标题：《jetty 介绍以及小例子- aspirant - 博客园》
来源：https://www.cnblogs.com/aspirant/p/9445542.html
12. 相似度：0.41% 标题：《使用Jetty作为嵌入式服务器_服务器应用_Linux公社-Linux系统门户...》
来源：https://www.linuxidc.com/Linux/2013-07/86983.htm
13. 相似度：0.39% 标题：《《React Native 精解与实战》书籍连载「React 与 React N..._博客园》
来源：https://www.cnblogs.com/parry/p/rn_book_intro_react_and_reactnative.html
14. 相似度：0.37% 标题：《计算机集群技术概述word免费下载》
来源：http://ishare.iask.sina.com.cn/f/2YLHduQFD30.html
15. 相似度：0.33% 标题：《php开发的B/S系统中,用户登录后申请某个权限,不重新登陆就能生效,...》
来源：https://ask.csdn.net/questions/350857
16. 相似度：0.29% 标题：《LinkedBlockingQueue生产者消费者问题?-CSDN论坛》
来源：https://bbs.csdn.net/topics/390589107
17. 相似度：0.29% 标题：《用阻塞队列LinkedBlockingQueue实现生产者消费者先进先出 孤独...》
来源：https://my.oschina.net/chengzz/blog/910112
18. 相似度：0.27% 标题：《阿尤博客 | 记录一个程序员技术成长经历》
来源：https://www.cnyou.org/
19. 相似度：0.25% 标题：《煽情的毕业感言精选3篇.doc》
来源：https://max.book118.com/html/2016/1208/69685101.shtm
20. 相似度：0.24% 标题：《Ambari架构及安装 - 大数据和AI躺过的坑 - 博客园》
来源：https://www.cnblogs.com/zlslch/p/6116264.html
21. 相似度：0.24% 标题：《Servlet 容器 Jetty 9.3.23 发布,Jetty 9.2 生命周期结..._开源中国》
来源：https://www.oschina.net/news/94015/jetty-9-3-23-released
22. 相似度：0.23% 标题：《大学毕业感言100字doc下载》
来源：http://ishare.iask.sina.com.cn/f/35EcdU5ai0b.html
23. 相似度：0.21% 标题：《Django之(URL)路由系统- 予沫笙- 博客园》
来源：https://www.cnblogs.com/yumo1627129/p/7718783.html
24. 相似度：0.18% 标题：《机构升格的请示,机构升格的请示资讯 - 高顿资讯搜索 -第1页》
来源：https://www.gaodun.com/news/q_10819794_p1/
25. 相似度：0.18% 标题：《毕业感悟 快资讯》
来源：http://www.360kuai.com/pc/98d2b6df63f34dc26?cota=4&kuai_so=1&tj_url=so_rec
26. 相似度：0.12% 标题：《分布式与集群的区别是什么?_知乎》
来源：https://www.zhihu.com/question/20004877
27. 相似度：0.11% 标题：《如何反驳 读大学时,获取知识的目的是为了完善人格?(PS:..._百度知道》
来源：https://zhidao.baidu.com/question/201350427683415645.html
28. 相似度：0.11% 标题：《Django路由系统 - 让我们忘了那片海 - 博客园》
来源：https://www.cnblogs.com/chenchao1990/p/5278900.html
29. 相似度：0.10% 标题：《IEEE TRANSACTIONS ON MOBILE COMPUTING IF=4.098,期刊投稿经验...》
来源：http://www.letpub.com.cn/index.php?page=journalapp&view=detail&journalid=3403
30. 相似度：0.09% 标题：《handleMessage 重复触发-CSDN论坛》
来源：https://bbs.csdn.net/topics/390322325
31. 相似度：0.06% 标题：《一个三层分布式计算网格任务调度系统》
来源：http://www.360doc.com/content/12/0519/10/9579107_212060503.shtml
32. 相似度：0.06% 标题：《海洋资讯(460)| 首艘国产豪华邮轮吨位比泰坦尼克大3倍 中国制造...》
来源：http://www.sohu.com/a/211160326_100013296
33. 相似度：0.05% 标题：《请教各位大佬,我在session中存了一个map,map的形式为Ma..._CSDN论坛》
来源：https://bbs.csdn.net/topics/392258956
34. 相似度：0.04% 标题：《MongoDB和Redis-NoSQL数据库-文档型-内存型 - 伊周 - 博客园》
来源：https://www.cnblogs.com/HANYI7399/p/5455129.html
35. 相似度：0.03% 标题：《 多线程 run()方法中异常处理- - 阿童沐 - 博客园》

来源：https://www.cnblogs.com/Tony-Mu/articles/2498653.html

全文简明报告

学 号 2015201109
密 级 20152011
哈尔滨工程大学学士学位论文
作业调度平台中心控制系统的设计与实现
院（系）名 称：软件学院
专 业 名 称：软件工程
学 生 姓 名：李佰波
指 导 教 师：马志强
哈尔滨工程大学
2019年6
2955578-7716作业调度平台中心控制系统的设计与实现
0作业调度平台中心控制系统的设计与实现
学 号
密 级
作业调度平台中心控制系统的设计与实现
Design and Implementation of Central Control System for Job Scheduling Platform
学生姓名：李佰波
所在学院：软件学院
所在专业：软件工程
指导教师：马志强
职称：
所在单位：哈尔滨工程大学
论文提交日期：2019年6月
论文答辩日期：2019年6月
学位授予单位：哈尔滨工程大学

{89%：摘 要中国数值水池网站创立初期肯定不可能拥有庞大的用户流量和海量数据，而是一步步演变其自身架构满足自身业务，因为系统的做大往往与业务的做大呈正比的。 }{99%：对于一个刚上线的项目，我们往往会将Web服务器、文件服务器和数据库全都部署在同一台物理服务器上。 }{90%：单机结构的优点在于使用简单， }{97%：配置成本低，数据共享程度高，一致性好， }{ 66%：而当业务发展，计算任务激增，用户量不断上升，系统瓶颈便开始暴露出来， }{100%：需要解决的主要问题是提升系统的并行处理能力， }

{ 58%：本课题旨在设计并实现一套面向集群的核心作业调度管理系统， }该系统能够完成求解程序的分发，同时提供多种调度策略，能够综合求解器的特点及各节点实际运行状态，合理地进行任务的调度。{ 73%：从而降低单机系统负载，一边支撑更多的用户访问操作以及计算求解任务量， }{93%：面向集群的作业调度技术可以将多台独立的服务器通过网络相互连接组合起来，形成一个有效整体对外提供服务，使用集群的意义在于其目的收益高于所付出的实际成本和代价。 }

本课题研究工作结合集群技术与调度策略算法，为解决系统自身的健壮性，高可用性，高并发以及负载均衡提供了新的技术手段。{ 58%：大量的实验结果证明了本文所提供方法的实用性和有效性。 }

关键词：作业调度；集群；高可用；高并发；负载均衡

ABSTRACTAt the beginning of the establishment of the China Digital Pool website, it is impossible

to have huge user traffic and massive data, but to gradually evolve its own architecture to meet its own business, because the system is often bigger than the business. For a project that is just online, we tend to deploy the web server, file server, and database all on the same physical server. The advantage of the stand-alone structure is that it is simple to use, low in configuration cost, high in data sharing, and good in consistency. When the business develops, the number of computing tasks increases,{ 60% : and the number of users continues to rise, } the system bottleneck begins to be exposed. The main problem to be solved is to upgrade the system. Parallel processing power,

This project aims to design and implement a cluster-oriented core job scheduling management system. The system can complete the distribution of the solver and provide various scheduling strategies. It can comprehensively solve the characteristics of the solver and the actual running status of each node. Scheduling of tasks. Thereby reducing the load of the single system, while supporting more user access operations and calculating the solution task, the cluster-oriented job scheduling technology can connect multiple independent servers through the network to form an effective overall external service, using the cluster. The significance is that the purpose of the benefits is higher than the actual cost and cost.

This research work combines cluster technology and scheduling strategy algorithms to provide new technical means for solving the system's own robustness, high availability, high concurrency and load balancing.{ 62% : A large number of experimental results prove the practicability and effectiveness of the methods provided in this paper. }

Keywords: job scheduling; cluster; high availability; high concurrency; load balancing

目录

{ 74% : 第1章 绪论1.1课题背景中国数值水池虚拟实验系统时利用先进的水动力学理论模型和精细数值算法，融合了专家的指挥，系统化编制的高效计算软件，经物理实验验证后，结合先进的计算机和互联网条件，为全球的行业用户提供在不同的海洋环境中，进行船舶与海洋结构流体动力响应过程的虚拟实验，满足船舶与海洋工程领域研究，设计及工程应用的要求。 }随着众多的虚拟实验集成以及用户数的激增，单节点服务器在进行虚拟实验数值计算时会产生性能瓶颈，{ 68% : 一台服务器已经不能满足应用的需求， }而需要更多的服务器集群来支撑庞大的计算量。开发数值水池作业调度系统将优化求解器计算任务在服务器组之

间的分配，{ 65% : 消除了服务器之间的负载不均衡， }{ 62% : 从而提高主系统的反应速度与总体性能。 }

1.2课题目的和意义数值水池虚拟实验网站创立初期采用的单机节点服务器虽然使用简单，{97% : 配置成本低，数据共享程度高，一致性好， }但是由于随着虚拟试验集成的范围越来越广，用户数量逐渐上升，求解计算任务激增，数据量呈现海量增长，{ 67% : 将所有的业务项目如（web服务器，文件服务器，数据库）全部部署服务到一台服务器上，所有的请求业务都由这台服务器处理。 }{100% : 显然，当业务增长到一定程度的时候，服务器的硬件会无法满足业务需求。 }{ 63% : 自然而然不可能支撑庞大的用户流量和海量数据，必须演变其自身架构。 }需要对网站做出如下调整：{100% : 独立部署，避免不同的系统之间相互争夺共享资源（比如CPU、内存、磁盘等）； }{ 61% : Web服务器集群，实现可伸缩性； }{100% : 部署分布式缓存系统，使查询操作尽可能在缓存命中； }{100% : 数据库实施读写分离，实现HA（High Availability）架构。 }

{ 55% : 本课题旨在设计与实现一个面向集群的作业调度管理系统平台， }在千万量级的计算任务中，由于存在大量求解任务请求并发访问，需要降低用户请求响应延时，需要合理设计调度策略，解决集群服务器组之间的计算任务合理分配，并对计算任务进行管理。同时完成各类虚拟实验求解器的按需部署及自动同步机制，给出个节点求解器计算状态。实现任务的集群分发，处理异常，保证可靠性。以及维护各类信息的日志，用于后续查询及决策。

在实际应用中，系统管理员可以通过管理页面查询计算集群节点资源状态，并查看每个计算节点的计算任务分配与求解器运行状态，可通过日志系统对求解任务进行回溯，远程一键kill不合理的计算任务，方便数值水池开发人员对虚拟实验集成及网站开发的后续决策。

国内外研究现状在集群作业调度研究方面，{90% : 主要分为定时分片类作业调度系统和DAG工作流类作业调度系统。 }

（1）定时分片类作业调度系统

{97% : 第一，定时分片类系统的方向，重点定位于任务的分片执行场景，这类系统的代表包括：TBSchedule，SchedulerX，Elastic-job, Saturn。 }{100% : 这种功能定位的作业调度系统，其最早的需要来源和出发点往往是做一个分布式的Crontab / Quartz。 }{100% : 一开始各个业务方八仙过海，自己玩自己的单机定时任务，然后，随着业务的增长，各种定时任务越来越多，分散管理的代价越来越高。 }{100% : 再加上有些业务随着数据量的增长，为了提高运行效率，也需要以分布式的方式在多台机器上并发执行。 }{100% : 这时候，分布式分片调度系统也就孕育而生了。 }{100% : 这类系统的实际应用场景，往往和日常维护工作或需要定时执行的业务逻辑有一定关联。 }{98% : 比如需要定时批量清理一批机器的磁盘空间，需要定时生成一批商品清单，需要定时批量对一批数据建立索引，需要定时对一批用户发送推送通知等等。 }{97% : 这类系统的核心目标基本上就是两点：对作业分片逻辑的支持：将一个大的任务拆成多个小任务分配到不同的服务器上执行， 难点在于要做到不漏，不重，保证负载平衡，节点崩溃时自动进行任务迁移等；高可用的精确定时触发要求：因为往往涉及到实际业务流程的及时性和准确性，所以通常需要保证任务触发的强实时和可靠性。 }所以，{83% : 负载均衡，弹性扩容，状态同步和失效转移通常是这类调度系统在架构设计时重点考虑的特性。 }第二，从接入方案和流程上来说，{100% : 因为要支持分片逻辑，要支持失效转移等，这类调度系统，对所调度的任务通常都是有侵入性要求的。 }太注重算法的时间复杂度，分析千量级节点的图谱时需要消耗过多的时间。{98% : 第三，从触发实现逻辑的角度来说，为了在海量任务的情况下，保证严格精确定时触发，这类调度系统有一大半，其定时触发逻辑，实际上是由执行节点自身在本地触发的，也就是说要求作业或守护进程处于运行状态，向服务端注册作业，服务端分配分片信息和定时逻辑给到客户端，但定时的触发，是由客户端库函数封装的如Quartz等定时逻辑来实际执行触发的。 }{100% : 这样做的首要目的当然是为了保证触发的精度和效率，降低服务端负载，此外如果服务端短时间内挂掉，只要作业配置保持不变，作业还是能够在客户端正常触发的。 }{100% : 也有些系统，比如SchedulerX，是采用服务端触发逻辑的。 }{100% : 这对服务端的要求就高了很多，因为这时候，服务端不光要协调分片逻辑，还要维护触发队列。 }{91% : 所以服务端触发的系统，首先要保证服务端的高可用， }其次

还要保障性能，因此，通常都是采用集群方案

(2) DAG工作流类作业调度系统

这一类系统的方向，{100% : 重点定位于任务的调度依赖关系的正确处理，分片执行的逻辑通常不是系统关注的核心，或者不是系统核心流程的关键组成部分，如果某些任务真的关注分片逻辑，往往交给后端集群（比如MR任务自带分片能力）或者具体类型的任务执行后端去实现。 }{100% : DAG工作流类调度系统所服务的往往是作业繁多，作业之间的流程依赖比较复杂的场景，比如大数据开发平台的离线数仓报表处理业务，从数据采集，清洗，到各个层级的报表的汇总运算，到最后数据导出到外部业务系统，一个完整的业务流程，可能涉及到成百上千个相互交叉依赖关联的作业。 }{96% : 所以DAG工

作流类调度系统关注的重点，通常会包括：第一：足够丰富和灵活的依赖触发机制：比如时间触发任务，依赖触发任务，混合触发任务；}{96%：而依赖触发自身，可能还要考虑，多亲依赖，长短周期依赖（比如小时任务依赖天任务，或者反过来），依赖范围判定（比如所依赖任务最后一次成功就可以触发下游，还是过去一个星期的所有任务都成功才可以触发下游），自身历史任务依赖，串并行触发机制等等。}第二：作业的计划，{84%：变更和执行流水的管理和同步。 }第三：{98%：任务的优先级管理，业务隔离，权限管理等， }{100%：在定时分片类调度系统中，通常情况下，具体执行端的业务的隔离很多情况下是天然的，注册了特定业务的节点才会去执行特定的任务。 }{99%：然后，加上业务链路一般都比较短，以及强实时性要求，所以对优先级的管理通常要求也不高，基本靠资源隔离来实现资源的可用，不太存在竞争资源的问题，权限管理也同理； }{88%：而在DAG工作流类调度系统中，往往一大批作业共享资源执行，所以优先级，负载隔离，和权限管控的问题也就突显出来。 }{95%：第三：各种特殊流程的处理，比如暂停任务，重刷历史数据，人工标注失败／成功，临时任务和周期任务的协同等等； }{97%：这类需求，本质上也是因为业务流程的复杂性带来的，比如业务逻辑变更啦，脚本写错，上游数据有问题，下游系统挂掉等等，而业务之间的网状关联性，导致处理问题时需要考虑的因素很多，也就要求处理的手段要足够灵活强大。 }第四：{83%：完备的监控报警通知机制，最简单的比如，任务失败报警，超时报警，再进一步，流量负载监控，业务进度监控和预测，如果做的再完善一点，还可以包括业务健康度监控分析，性能优化建议和问题诊断专家系统等。 }

1.4 主要研究内容本课题旨在完成一个面向集群的作业调度中心控制系统的设计与实现，将网站架构从单机节点过渡到集群节点，并保证数值水池虚拟试验计算任务在各集群服务器组中进行合理分配和调度。

作业调度平台中心控制系统是一种反向代理机制。中心控制节点提供反向代理服务，根据客户端计算求解任务的请求，将计算任务和求解器分配到与其关系的一组或多组后端服务器，实现各种虚拟实验服务器簇的求解分离。中心控制节点会监控和管理计算任务，以实现后端虚拟实验服务器簇与中心控制节点反向代理服务器之间的求解器同步以及资源状态的收集。

实验与应用是作业调度平台中心控制系统有效性的验证。分别使用不同的用户以及虚拟实验计算任务量和服务器集群分别验证计算任务的合理调度和分发以及求解器自动同步机制，通过应用体现其实际价值。

{ 55%：1.5 论文结构安排本篇论文主要分为5章， }从面向集群作业调度平台的背景、研究的意义以及研究现状，{ 59%：到作业调度平台中心控制系统的设计与实现， }以及多种调度策略算法和求解器同步机制的验证与应用等方面进行论述。

{ 65%：第1章主要论述本课题的背景， }研究的意义目的、国内外的对本课题的研究现状以及本课题的主要内容方面做大体的说明。

第2章主要论述求解器按需部署及自动同步机制的实现，简要介绍求解器部署的需求及自动同步机制算法的设计。

第3章是重点论述集群各计算节点状态收集的实现

第4章重点论述多种调度策略算法的设计与实现，先论述选择合适的调度策略，然后将计算任务进行合理分配。

第5章是实验与应用，验证多种调度策略算法和求解器同步机制

{ 60%：最后是结论、致谢和参考文献。 }

第2章 系统总体设计2.1技术介绍MongoDB数据库以及其分布式部署：{ 58%：mongoDB数据库是一种属于文档型的Nosql类型数据库， }{ 60%：数据模为Key-Value对应的键值对， }Value为结构化数据。文档可以有一个灵活的模式，{100%：这意味着集合中的文档可以有不同(或相同)集的字段。 }{97%：这在处理数据时提供了更大的灵活性， }{94%：使用户能够存储嵌套或多值字段，如数组、散列等。 }{97%：它使用二进制JSON存储数据， }{100%：JSON代表JavaScript对象符号。 }{94%：在当今的现代Web(以及XML)中，它是用于数据交换的标准格式， }{92%：是人类和机器可读的，这不仅是交换数据的好方法，也是存储数据的好方法。 }JSON支持字符串，数字，布尔值和数组等，{92%：JSON可以将所有相关的信息集中在一个地方，这提供了优秀的信息性能， }使文档的更新成为独立的。{98%：MongoDB以二进制编码的格式存储JSON文档。 }{100%：这被称为“BSON”。BSON数据模型是JSON数据模型的扩展形式。 }{100%：MongoDB的一个BSON文档的实现是快速的、可实现的，并且是轻量级的。 }{100%：它支持在其他数组中嵌入数组和对象，并允许MongoDB进入对象内部在顶级和嵌套的BSON键上构建索引并匹配对象。 }

Jetty服务器:jetty的易用性，可扩展性，可嵌入性，{97%：使得开发人员可以将Jetty容器实例化成一个对象，可以迅速为一些独立运行（stand-alone）的Java应用提供网络和web连接， }{ 68%：可以使jetty很容易的嵌入到应用程序中，而不需要程序为了使用它而修改， }相比于tomcat而言，它更加灵活，{ 61%：

更加轻量级，更满足分布式环境的需求，}这也是为什么在本项目中选择jetty作为服务器而不是tomcat的原因。

2.2 数据库相关表介绍本系统所涉及的项目主要由四张表，agents, dists, solvers, tasks。以下为每张数据表一条记录所对应的主要字段。agents表是存放计算节点相关属性的表。

表2.1：agents表主要字段含义

字段 类型 含义

_id hash string 计算节点编号

name string 名称

port integer 端口

osstring 操作系统

path string 计算结果存放路径

maxLoadsinteger 最大负载

createtimetimestap创建时间

status integer 节点状态

loads integer 节点当前负载

updateTimetimestap更新时间

solvers array 节点安装求解器列表

suspend bool 节点是否挂起

dists表是用来连接solver和agent的映射关系。

表2.2：dists表主要字段含义

字段 类型 含义

_id hash string 任务编号

agent hash string 计算节点编号

solver hash string 求解器 编号

version integer 求解器版本

status integer 任务状态

createtimetimestap任务创建时间

solvers表是用来存放求解器的相关属性。

表2.3：solvers表主要字段含义

字段 类型 含义

_id hash string 求解器编号

name string 求解器名称

label string 求解器描述

user hash string 用户编号

createTime timestap创建时间

executor string 执行文件

updateTime" timestap更新时间

relativeFilesarray 求解器相关文件

mode integer 发布模式

osstring 求解器运行的操作系统

queue integer 排队方式

loader string 加载器
params string 求解器执行参数
file string 求解器指定版本文件
version integer 求解器版本
tasks表用来存放求解器任务记录
表2.4：tasks表主要字段含义
字段 类型 含义
_id hash string 任务编号
solver hash string 求解器编号
solverNamestring 求解器名称
workingDirectorystring 存放路径
status integer 任务状态
params string 求解器参数
createTimetimestap任务创建时间
startTimetimestap任务启动时间
message string 日志信息
updateTimetimestap任务更新时间
finishTimetimestap任务结束时间
在DB类中，会将四张表初始化成ConfigurableCache类对象，方便项目中逻辑代码对数据库的查询，更新及删除操作。
表2.5：初始化相关表为ConfigurableCache对象
static public ConfigurableCache solver;
static public ConfigurableCache agent;
static public ConfigurableCache dist; // 连接 solver 和 agent
static public ConfigurableCache task;
task = Cache.getCache("task");
solver = Cache.getCache("solver");
agent = Cache.getCache("agent");
dist = Cache.getCache("dist");
2.3 系统结构设计本系统采用Jetty作为内置服务器，基于RESTFUL软件设计架构。众所周知，URL是web应用服务的路径，用户通过浏览器发送过来的任何请求都会被发送到一个指定的URL地址里，然后被响应。首先创建cn.edu.hrbeu.theweb.server.WebServer类，{ 61%：这个类的功能是使用jetty作为嵌入式服务器，}{ 78%：把jetty部署到web应用中。 }{89%：因为Jetty可以在java应用中像其他POJO一样被实例化，换句话说，以嵌入式的模式运行Jetty是将http模块放入应用程序中，而非部署程序到HTTP服务器。 }首先创建一个Server实例，在私有方法configureServer()里添加/配置Connectors，{ 72%：添加/配置Handlers / Contexts 以及 Servlets。 }然后在构造函数WebServer(int serverPort)中传入serverPort，以serverPort为端口启动和监听Jetty，这样就首先完成了Jetty的嵌入式实现。
Jetty服务器成功后读取etc/settings.json配置文件，该配置文件以json的格式配置了MongoDB数据库地址(DATABASE_ADDRESS)，数据库名称(DATABASE_NAME),虚拟实验数据存放的根路径(PROJECT_BASE_PATH)，上传文件临时存放路径(UPLOAD_TEMP_PATH)，求解器存放根路径(SOLVER_BASE_PATH)，身份标识(Role).
cn.edu.hrbeu.theweb.server.HTTP类有同样的静态字段，并且附带有初始化的默认值。
在用户登陆系统验证身份成功后，通过LeaderService.init()启动Leader服务。系统的初始化流程图如下：

图 2.1 系统初始化流程图

（1）读取配置文件

读取etc/settings.json文件，若配置文件为空，则退出并结束。

（2）设置HTTP类的各字段值

根据配置文件中的键值对(key:value)设置相应的HTTP类中各字段值，如果某个字段值为空，则设置为相应的HTTP类字段缺省值。

HTTP.ROLE字段验证

HTTP.ROLE字段的缺省值为"Leader/Agent", 如果包含leader和agent中任何一项，则进行下一步。

（4）连接用户信息数据库2.4用户登录和超时注销当用户通过浏览器登录系统时，{ 60%：系统会给每个用户分配一个Session, 如果用户用同一个session访问系统的话，系统会将其识别出来，不应该在切换网页之后，系统就不识别该用户了。 }要实现这个功能，可以在后台逻辑中专门新建一个USER类，用来处理超时用户注销。该类继承自java.lang.Thread。{ 58%：将所有用户的session放在map数据结构中， }{ 63%：将一个用户和其相对应的session绑定起来，通过维护该SESSION_MAP可以实现对该用户的管理。 }该类中声明两个变量，LOOP_TIME_OUT和IDLE_TIME_OUT，LOOP_TIME_OUT所设定的时间间隔为60000毫秒，即1分钟，该时间间隔表示线程需要每隔相同的LOOP_TIME_OUT时间间隔检查一次SESSION_MAP；

IDLE_TIME_OUT所设定的时间为60000 * 60毫秒即一小时，如果用户无操作时间大于IDLE_TIME_OUT，则从SESSION_MAP中将其sessionId和用户对应关系移除，即注销了超时用户。

用户登录之后，在SESSION_MAP中添加用户和SessionID的对应关系

表2.6：SESSION_MAP中添加对应关系

```
public static String ADD_LOGIN_USER(Document user) {
String sessionId = M.getRandomString(20);
return ADD_LOGIN_USER(sessionId, user);
}
public static String ADD_LOGIN_USER(String sessionId, Document user) {
synchronized(SESSION_MAP) {
user.put("lastAccessTime", System.currentTimeMillis());
SESSION_MAP.put(sessionId, user);
M.trace("addLoginUser > session size = ", SESSION_MAP.size());
return sessionId;
}
}
```

{ 57%：在映射sessionId和用户的关系时， }在User文档中可以加入” lastAccessTime”用来记录该用户最后一次接入时间，用户无操作时间即为当前时间currentTime减去lastAccessTime，如果该时间间隔大于IDLE_TIME_OUT, 则注销该超时用户。

表2.7：注销超时用户

```
for (String session : SESSION_MAP.keySet()) {
Document user = SESSION_MAP.get(session);
long lastAccessTime = user.getLong("lastAccessTime");
long idle = System.currentTimeMillis() - lastAccessTime;
if (idle > IDLE_TIME_OUT) {
SESSION_MAP.remove(session);
}
}
```

{ 58% : 2.5 路由机制与视图渲染本系统的路由机制就是使处理数据的函数与请求的URL建立映射关系。 }使请求到来之后，根据cn.edu.hrbeu.theweb.server/目录下的RootResource，StaticResource，SolverFileResource，FileResource，LeaderFeedbackResource，LeaderServiceResource类里的正则表达式及注解条目，{93% : 去查找与请求相对应的处理方法，从而返回给客户端http页面数据。 }

相应的操作方法都加有@path,@post,@get,@produce注解。

{ 61% : @Path注释的值是表示在Java类将被承载的相对URI的路径：例如，/ HelloWorld的, }还可以在URI中嵌入变量以生成URI路径模板。{ 67% : 例如，可以询问用户的名称，并将其作为URI中的变量传递给应用程序： / helloworld / {username}； }

{100% : @POST注解是请求方法指示符， }并对应于类似命名的HTTP方法。{ 62% : 使用此请求方法指示符注释的Java方法将处理HTTP POST请求。 }{84% : 资源的行为由资源响应的HTTP方法确定； }

{ 63% : @Produces注释用于指定MIME媒体类型表示的资源可以产生和发送回客户端：例如，“text / plain的” ； }

{ 59% : @Consumes注释用于指定MIME媒体类型表示资源可以消耗这是由客户端发送的； }@GET注解是请求方法指示符，并对应于类似命名的HTTP方法，{ 61% : 使用此请求方法指示符注释的Java方法将处理HTTP GET请求， }{84% : 资源的行为由资源响应的HTTP方法确定； }

{ 60% : @PathParam 注释是一个类型参数，可以提取的资源类的使用， }{ 67% : URI路径参数从请求URI中提取，参数名称对应于@Path类级别注释中指定的URI路径模板变量名称。 } @path注解可以加入正则表达式用来匹配请求路径映射。例如：@Path("solver/{id: [a-zA-Z_0-9]*}/file/download/{fileLink: .*}")是用来匹配求解器文件下载请求映射的函数，@Path("solver/{id: [a-zA-Z_0-9]*}/file/delete/{fileLink: .*}")是用来匹配删除求解器文件请求映射的函数，@Path("/css/{subResources:.*}")是用来匹配静态资源目录下css子目录下的层叠样式表请求映射的函数。系统总的路由机制图与视图渲染如下所示：

图 2.2 路由机制图

首先客户端发来的http请求的根目录 / 可以映射到所有的Resource类（RootResource， StaticResource，SolverFileResource， FileResource，LeaderFeedbackResource，LeaderServiceResource）。

（1） RootResource类相当于一个一级路由，主要处理用户信息操作请求，比如用户登陆处理函数loginPost()，由/ login路径进入；用户注销处理函数logout()，由/ logout路径进入；用户注册处理函数regist()，由/regist路径进入。以及doApi(String tt, String op, Document input, LoginedUser lu)函数用来处理系统总的api请求。其中tt是targetTable的缩写，在一级路由时被识别，即要操作的目标表名；op是一些对特殊类的具体操作指令，在二级路由被识别, 如果在具体类的二级路由中没有被识别，则默认跳转到DB中的api（ ）路由函数进行处理，Configurable中的api路由函数是对数据库表记录的具体操作的封装。

Solvers类二级路由如下:

表2.8：solvers类二级路由

op(操作) 处理函数 含义

list __SolverList获取求解器列表

delete solverDelete求解器删除

distribute solverDistribute求解器分发

update solverUpdate求解器更新

file-list solverFileList获取求解器文件列表

file-get solverFileGet获取求解器文件file-delete solverFileDelete求解器文件删除

file-appoint solverFileAppoint求解器文件指定

run run 运行求解器

Projects类二级路由如下:

表2.9：projects类二级路由

op(操作) 处理函数 含义

file-list projectFileList显示项目目录

code projectCode显示项目记录

Taks类二级路由如下：
表2.10：tasks类二级路由
op(操作) 处理函数 含义
kill kill 结束任务
Agents类二级路由如下：
表2.11：agents类二级路由
op(操作) 处理函数 含义
infoInsert__AgentInfoInsert插入节点信息
list __AgentList获取节点列表
get __AgentGet获取节点
insert __AgentInsert增加节点
delete __AgentDelete删除节点
deletemany删除多个节点
distribute __AgentDistribute发布节点
solversList__AgentSolversList获取节点已安装求解器列表
solvers-list solversSelect__AgentSolversSelect节点选定求解器
solvers-select solversInsert__AgentSolversInsert节点部署求解器
solvers-insert solversDelete__AgentSolversDelete节点删除求解器
solvers-delete __AgentSolversDelete节点删除求解器
Configurable类路由如下:
表2.12：configurable类二级路由
op(操作) 处理函数 含义
page pageItems获得某page数据
insert insertItem插入一项数据
get getItem获得一项数据
update updateItem更新一项数据
upsertupsertItem修改或添加一项数据
set setItem修改一项数据
array-insert arrayInsertItem以数组方式插入一系列数据
array-update arrayUpdateItem以数组方式更新一系列数据
array-delete arrayDeleteItems以数组方式删除一系列数据
vector-push vectorPushItems以向量方式插入一系列数据
vector-pull vectorPullItems以向量方式删除一系列数据
vector-update vectorUpdateItem以向量方式更新一系列数据
vector-copy vectorCopyItem以向量方式复制一系列数据
push pushItem插入一项数据
pushlistpushList插入列表数据
pulllistpullList删除列表数据
pull pullItem删除一项数据
list getList获取列表数据

foreign-list getForeignList获取从表数列表数据

list-as-foreign getListAsForeign根据指定主表及其字段获取从该表列表

delete deleteItem删除一项数据

deletemanydeleteMany删除多项数据

count count 返回记录计数器

distinct-count distinctCount返回非重复记录计数器

quick-search quickSearch快速查询相关数据

distinct distinctString对相关字段去重

lookup lookup 查询并对记录丢失数据处理

clear clear 清空多项数据

aggregate aggregate 聚合数据

top getTop查询基本数据

图 2.3 RootResource类路由机制与系统视图渲染图

（2）StaticResource类主要用来处理静态资源请求映射的URL.{ 78% : 如javascript脚本， }image图片以及css层叠样式表。

（3）SolverFileResource类主要用来处理与求解器文件有关的请求操作，如求解器文件的上传，下载，删除，打开，以及求解器文件属性查询。

（4）FileResource类主要用来处理与文件相关的请求操作。如文件的上传，下载，删除，以及文件属性查询。

(5) LeaderFeedbackResource类主要用来处理中心控制节点LEADER 对 AGENT 内部反馈服务接口，处理AGENT api请求和 AGENT文件同步。

(6) 对所有路由函数的转发处理，大部分操作归根结底转化为对MongoDB数据库的操作。

（7）前端浏览器界面是react组件，react是Mode-Driven View的实现，{94% : 可以替代过去 JavaScript 代码直接操作前端真实 DOM，而是完全通过 state 以及 props 的变更引起页面 DOM 的变更，这比 jQuery 等框架那样进行大量的 DOM 查找与操作来得简单、高效的多。 }优点：组件化 ，易复用解耦 ，数据控制视图。通过加载数据控制不同的视图渲染在浏览器端界面。

2.6功能模块介绍：本系统主要模块有求解器模块，任务模块，节点管理模块，日志模块。求解器模块分为求解器按需部署，求解器自动同步机制，求解器状态获取；任务模块分为集群任务分发，多种调度策略，数据统计子模块；节点管理模块分为维持心跳，发送指令，接收反馈，节点资源状态获取模块；日志模块分为日志维护与异常处理子模块。

图 2.4 系统功能模块图

2.7本章小结：本章讨论系统的总体架构，包括采用的相关技术，如MongoDB数据库和Jetty服务器，{ 58% : 描述了为什么使用该文档型Nosql数据库而不是传统的关系型数据库， }以及jetty相比于tomcat是本项目更适合采用的服务器。介绍相关主要数据库表(solvers, tasks, agents, dists)相关字段的含义，系统结构设计，用户登录和超时注销，论述采用的路由机制与视图渲染方法，以及展示了系统主要的功能模块。

第3章 求解器按需部署及自动同步机制3.1 需求分析当网站服务器为单机架构时，对于求解器的部署更新往往采用人为手动复制进行版本的安装迭代，仅仅能应付初期求解器版本迭代较慢的时期。当网站架构转化为分布式集群架构时，集群中有多个计算节点时，采用人工部署安装方式效率低下，可能会导致版本错误以及文件复制遗漏等情况。如果求解器部署出现错误，用户在进行计算任务求解时会导致计算结果错误，对于计算耗时耗资源的一类求解器任务将会大量消耗服务器资源并且无法得出正确计算结果，会严重影响用户体验以及虚拟实验集成项目的进度开发，后期开发人员对系统进行维护时必将是个噩梦，为了减轻系统维护人员负担，增强用户体验以及提高项目的健壮性，迫切需要实现求解器模块相关功能，实现求解器的自动化部署。

3.2 求解器按需部署SolverFileResouce类专门用来处理与solver求解器相关的请求，用户请求通过根路径/可以直接跳转到该类。

表 3.1 SolverFileResource函数功能表

请求路径 函数名称 返回值类型含义

createSourceFilejava.io.File创建求解器源文件

getSourceFilejava.io.File获取求解器源文件

getHomeDirectoryjava.io.File获取求解器安装根路径

solver/{id}/file/delete/{fileLink: .*} fileDeletejava.lang.String删除求解器文件
solver/{id}/file/download/{fileLink: .*} fileDownloadjavax.ws.rs.core.Response求解器文件下载

solver/{id}/file/open/{fileLink: .*} fileOpenjavax.ws.rs.core.Response打开求解器文件
solver/{id}/file/info/{fileLink: .*} fileInfojava.lang.String获取求解器文件属性

solver/{id}/file/upload fileUploadjava.lang.String上传求解器文件Solvers类里包含了对求解器的基本操作

表3.2：Solvers类函数功能表

函数名称 返回值类型含义

router org.bson.Document创建求解器源文件

__SolverListorg.bson.Document获取求解器源文件

getRelateAgentListjava.util.List获得与 solver 相关 agent 列表

solverUpdateorg.bson.Document求解器更新

solverDistributeorg.bson.Document求解器发布solverFileAppointorg.bson.Document求解器指定

solverDeleteorg.bson.Document求解器删除

getComandLinejava.lang.String获取求解器启动命令

run org.bson.Document启动求解器

solverFileGetorg.bson.Document获取求解器文件listSolverFilejava.util.List<Document> 获取指定目录下所有求解器

solverFileListorg.bson.Document显示求解器所有文件列表

solverFileDeleteorg.bson.Document求解器文件删除

createSourceFilejava.io.File创建求解器源文件

getSourceFilejava.io.File获取求解器源文件

getHomeDirectoryjava.io.File获取求解器安装根路径

管理员用户可以根据前端界面对求解器属性进行修改，例如新建/删除求解器。新建求解器后可以编辑求解器属性，如求解器名称，求解器描述，选择发布模式（包括集群发布和仅本地发布），求解器适合运行的操作系统，版本号，加载器(java, python)等，启动程序，启动参数，排队方式(标准队列，快速队列，不排队)。集群发布表示将该类求解器任务在所有求解器节点上进行发布，根据后续的调度策略选择最优计算节点进行计算；本地发布表示将该类求解器任务放在本地中心控制节点进行求解计算而不进行集群发布，减少了根据调度策略进行选择最优节点进行计算的步骤。求解器的排队方式有三种：标准队列表示求解器在集群发布时进行所有求解任务入队列操作，根据相应的调度策略进行选择最优计算节点的过程。选择快速队列则表示该类求解器任务具有优先级，例如根据以往每类求解器运行时间长短，可以将能够快速得到计算结果的求解器任务选择快速队列，目前该系统支持的快速队列节点为本地节点，省去了调度策略步骤，可以更快的响应用户实验结果的请求。不排队方式也是放在本地节点进行计算求解的，可以对应具有特殊需求的一类求解器任务。

表3.3：求解器按需部署属性表

求解器名称 描述 发布模式 操作系统 版本 加载器 启动程序 创建时间

K0304船舶快速性预报 集群发布 win 5 srm.exe 2017/11/5

K0401激振力预报 集群发布 win 1 Bulk_Carrier_prediction.exe 2018/6/29

K0402面元法空泡激振力 集群发布 win 2 K0402AllRun.exe 2018/11/14

K0502船舶运动 集群发布 win 5 K0502AllRun.exe 2018/6/29

K0503波浪增阻实验 集群发布 win 2 K0503AllRun.exe 2017/7/26

K0602快速自航模 集群发布 win 3 K0602AllRun.exe 2018/9/26

K0603船舶操纵运动 集群发布 win 6 ShipMaDBase.exe 2017/11/5

K0701大尺度海洋环境 集群发布 win 1 run.exe 2017/11/5

K0703中小尺度 集群发布 win 2 K0703AllRun.exe 2018/12/5

K0803作业海况 集群发布 win 6 K0803AllRun.exe 2017/10/13

K0803B作业海况耦合分析 集群发布 win 2 Coupled_IntelV1.exe 2017/11/21

K0901涡激振动 集群发布 win 1 dyn_xml.exe 2017/11/13

求解器按需部署属性对应MongoDB的数据库字段如下：

{
"_id" : "59e041ddf589d45478a2d960",
"name" : "K0803",
"label" : "作业海况",
"executor" : "K0803AllRun.exe",
"user" : "5978ab95f589d41b9cbab06d",
"createTime" : NumberLong(1507869149575),
"file" : "K0803A_20190408.zip",
"updateTime" : NumberLong(1554697672166),
"os" : "windows",
"mode" : 0,
"queue" : 0,
"version" : 4,
"loader" : "",
"params" : ""
}

该json字段表示名称为K0803的求解器其部署属性，求解器执行文件为K0803AllRun.exe，user表示关联创建者ID， file指定求解器版本文件名称，createTime和updateTime分别是创建求解器和更新求解器的两个时间戳，os表示求解器运行的操作系统环境，mode为0表示集群发布，queue为0表示该求解器类型计算任务在分发时按标准队列排队，version表示求解器版本， loader和params为空表示求解器运行时不要加载器和启动参数。

在前端界面中，用户编辑好每一个求解器属性后可进入求解器信息详细界面，左侧显示求解器属性和执行参数等已编辑信息，右侧分别显示求解器执行文件和该类求解器对应的计算任务列表。求解器文件支持上传和刷新操作，目前求解器上传文件仅支持zip格式，求解器执行文件列表将显示最新版本求解器压缩文件及其相关文件，对每个版本的求解器版本压缩文件有两个操作：指定和删除。如果一个版本的求解器版本压缩文件被指定后，后续该类求解计算任务都会根据此版本的求解器进行求解计算。管理员每上传一个求解器执行文件，则新求解器执行文件的版本号加1，对应数据库相应字段version增1。

当求解器按需部署时，需要在计算节点界面安装求解器列表面板选择添加未安装求解器，选定好需要在本节点部署的求解器后，计算节点agent会向控制leader发出“syn”同步指令，调用相关方法，由求解器自动同步机制实现控制节点到计算节点的求解器版本迭代安装。

3.3 求解器自动同步机制集群中每个计算节点详细界面可以显示该节点已经安装的求解器列表，管理员可以选择在该列表中添加求解器，点击添加求解器后，会显示控制节点求解器中央仓库中所有最新版本的求解器，管理员可以根据复选框添加相应的求解器。之后求解器会根据相应的自动同步机制进行同步。例如计算节点E206，IP地址为192.168.1.216的节点上已经安装求解器列表如下：

表3.4：E206计算节点已安装求解器列表

求解器名称 版本 描述 操作系统 执行状态

K03045 船舶快速性预报 MS Windows 同步成功

K04011 激振力预报 MS Windows 同步成功

K06023 快速自航模 MS Windows 同步成功

K06036 船舶操纵运动 MS Windows 同步成功

K07011 大尺度海洋环境 MS Windows 同步成功

K09011 涡激振动 MS Windows 同步成功

如果管理员需要在某个计算节点上添加相应的求解器，当复选未安装求解器列表中的求解器并点击确认时，相应的计算节点会向中心控制节点发出特定的http下载请求，从控制节点求解器中央仓库目录下载选中的求解器到计算节点本地求解器目录。接着在该计算节点相关联的求解器列表就可以看到刚刚安装的求解器了。求解器的同步以及启动状态在DEF类里定义，如下表所示：

表3.5：求解器同步及启动状态

状态 含义

_SOLVER_SYN_STATUS_CAN_NOT_SYN = 0 无法同步

_SOLVER_SYN_STATUS_WAITING = 1 等待启动

_SOLVER_SYN_STATUS_STARTED = 2 启动

_SOLVER_SYN_STATUS_FINISHED = 3 已同步成功

_SOLVER_SYN_LEADER_FILE_LOST = -1 无法连接上级

_SOLVER_SYN_STATUS_FILE_CAN_NOT_DOWNLOAD = -2 上级无法下载文件

_SOLVER_SYN_STATUS_FILE_CAN_NOT_UNZIP = -3 文件无法解压

3.4 求解器状态获取中心控制节点需要知道自己本地的求解器状态以及计算集群各节点上的求解器状态。

本地求解器状态：通过查询数据库相关字段获取。

集群求解器状态：计算节点维护一个AgentSolverFileSynKeeper实例线程，在求解器同步的每个阶段调用reportSolverStatus方法，向控制节点发送自身的求解器状态信息。

表3.6：集群节点求解器状态获取

```
void reportSolverStatus(String solverId, int version) {
Document message = new Document()
.append(DEF.OP, "solverSyn")
.append(DEF._SOLVER, solverId)
.append(DEF._VERSION, version);
this.feedbackKeeper.sendMessage(message);
}
```

3.5 异常处理（1）求解器更新操作solverUpdate函数需要处理的异常：用户无操作权限，参数错误缺少求解器ID，更新的value值格式错误，求解器不存在，更新失败。

（2）求解器发布操作solverDistribute函数需要处理的异常：用户无操作权限，参数错误，求解器不存在，操作系统未指定，未指定求解器文件，指定文件不存在，指定文件类型错误，发布失败。

（3）求解器文件指定操作solverFileAppoint函数需要处理的异常：用户无操作权限，参数错误，仅支持zip格式文件，求解器不存在，指定文件已经在使用中，指定文件不存在，指定文件类型错误。

（4）求解器删除操作solverDelete函数需要处理的异常：用户无操作权限，删除失败。

（5）获取求解器启动参数操作getComandLine函数需要处理的异常：未输入启动命令。

（6）求解器运行时操作run函数需要处理的异常：用户无操作权限，求解器不存在，求解器正在执行中不能重复启动。

（7）显示求解器列表操作__SolverList函数需要处理的异常：用户无操作权限，数据库访问错误。

3.6 本章小结本章论述作业调度平台中心控制系统求解器功能模块的需求分析，求解器的按需部署包括求解器属性的编辑，选择求解器运行的操作系统，发布模式，排队方式等。求解器的自动同步机制是当管理员在某个计算节点添加未安装求解器时，计算节点会自动向中心控制节点请求同步该求解器最新版本的可执行文件，以及与求解器相关操作函数的异常处理。

第4章 计算节点管理4.1 需求分析：中心控制节点要实现对计算节点进行管理，就必须要与下级计算节点进行通信，维持主从节点通信需要实现特定的心跳机制，控制节点发送消息到计算节点， 控制接待你接收计算节点消息时进行反馈。对于集群管理，一个必不可少的步骤就是集群监控，为什么需要集群监控？其实这个问题不言而喻，{91%：成百上千台机器在哪里工作，如果不知道它们的工作状态，岂不是一件很棘手的问题。 }{86%：还有，如果没有监控，怎么评价自己开发的程序是否高效，难道仅仅凭能够执行通过么？ }从架构上讲，{86%：主节点和从节点都会运行监控程序，主节点(以下称为leader)负责收集从节点数据并分析展示，从节点程序(以下称为agent)负责采集本节点的状态信息。 }从agent的角度来看数据采集的工作模式一般分为两种：主动模式和被动模式，{ 63%：主动模式即agent主动上报资源状态数据并汇报给leader， }被动模式即leader收集各agent资源状态数据(如当前计算节点的作业负载，{ 63%：CPU利用率，内存利用率等)。 }Leader所收集到的集群各节点资源状态数据对后续集群任务分发与调度策略的实现将产生十分重要的影响。

4.2心跳机制：NextAgentkeeper类用于保持对下级节点的状态处理，继承自java.lang.Thread，{ 60%：在每个线程体的run()方法中, } leader与agent进行维持心跳，通过轮询方式，判断消息队列queue头部取出的消息message是否为空，如果为空的话，则leader与agent 5秒进行一次消息的同步， 并且当每个agent上运行的任务错误量大于3时，则强制将该agent下线。

4.3 leader发送消息：NextAgentkeeper的每个实例线程内部实现了以LinkedBlockingQueue<Document>为数据结构的消息队列queue， leader可以向agent发送“syn”，“taskRun”，“taskKill”三类消息。每个消息类型时一个org.bson.Document对象。消息头包括包括TT即targetTable表示需要操作的目标数据库表；OP即Operation表示消息类型代表的具体操作；_PORT表示HTTP.SERVICE_PORT即连接的服务端口。

“syn”同步消息消息体为：{ 57%：_Agent代表自身节点信息； }_SOLVERS表示该agent已安装求解器列表信息；_TASKS表示和该agent有关联的任务信息包括正在执行任务，等待启动任务，完成启动任务。其消息格式图如下表所示:

表4.1： “syn”消息格式

TT OP _PORT _AGENT _SOLVERS _TASKS

“taskRun”任务启动消息消息体为_VALUE表示task的相关信息。

表4.2： “taskRun”消息格式

TT OP _PORT _VALUE

“taskKill”任务结束消息消息体为_ID表示需要结束任务的ID。

表4.3： “taskKill”消息格式

TT OP _PORT _ID

Leader发送消息时通过消息队列进行发送。其消息生产函数sendMessage(Document message), 通过对消息进行加锁，将消息添加到消息队列中并唤醒其他所有线程，主要代码如下：

表4.4 消息生产函数sendMessage原型

```
private void sendMessage(Document message) {
synchronized (queue) {
this.queue.add(message);
this.queue.notifyAll();
M.trace("NextHostKeeper recever message => ",
this.agentId, " ",
Docat.__doc_to_json(message));
}
}
```

消息消费过程在线程体run()方法中，方法体中会开启一个无线循环，通过queue.poll()方法从消息队列的头部取出消息，{ 71%：通过handleMessage(message)方法对消息进行处理， }主要代码如下：

表4.5 消息消费过程代码

```
try {
Document message = this.queue.poll();
if (message == null) {
if (System.currentTimeMillis() - this.lastTime > 5000) {
this.doSynSend();
}
break;
} else {
this.handleMessage(message);
}
} catch (Exception ex) {
ex.printStackTrace();
}
```

4.4 Leader接收消息反馈：Agent会通过主动模式用特定封装方法将自身的资源状态信息汇报给leader，leader在接收Agent消息后会进行消息反馈。LeaderFeedbackResource类是LEADER 对 AGENT 内部反馈服务接口。当agent发送消息给leader后，leader会将请求为json格式的消息进行分拆解析，相当于报文拆分，并处理其中的异常，其主要代码如下：

表4.6 Leader解析Agent消息

```
Document input = Document.parse(json);
op = input.getString(Cache.OPERATION_TAG);
Runtimes.throwIf(Strings.isEmptyStrictly(op), "格式错误，缺少OP!");
tt = input.getString(Cache.TARGET_TABLE_TAG);
```

Leader在解析Agent消息后，要进行响应，即消息反馈，其主要代码如下：

表4.7 Leader消息反馈

```
Document output = HTTP.okDocument().
append(Cache.TARGET_TABLE_TAG, tt).
append(Cache.OPERATION_TAG, op);
outJson = Cache.doc2json(output);
outJson = Cache.doc2json(new Document(Cache.RETURN_CODE_TAG,
ex.getMessage()));
return Response.status(200).entity(outJson).build();
```

通过以上代码不难发现，Leader进行消息反馈的消息格式如下表：其中OC代表RETURN_CODE_TAG即返回状态码。

表4.8 反馈消息格式

TT OP RC

4.6 Leader收集集群各agent的资源状态：以计算节点E208为例，在agents表结构中定义了如下字段：

```
{
"_id" : "5c9099d5f589d423d064a4cf",
```

"name" : "E208",
"ip" : "192.168.1.218",
"os" : "windows",
"port" : 10001,
"path" : "Y:/TANK/projects",
"maxLoads" : 4,
"updateTime" : NumberLong(1554791827884),
"status" : 1,
"loads" : 3,
"solvers" : [
"59fee826f589d42b7864def2",
"5b35f2bdf589d4116459bd5e",
"5beb788ff589d4312cc24eaf",
"5b35e93af589d4116459bd5d",
"5978ac60f589d41b9cbab06e",
"5bab4cabf589d4116459bd5f",
"59feeb4cf589d42b7864def3",
"59ff00b7f589d42b7864def4",
"5c0739cff589d4312cc24eb0",
"59e041ddf589d45478a2d960",
"5a1391ddf589d417b0735bb9",
"5a097e9bf589d41c54cfff68"
],
"suspend" : false
}

_id表示计算节点编号，name节点名称，ip地址，ip端口，os表示操作系统，path表示该计算节点存储求解任务计算结果的根路径。maxloads表示该计算节点设置的最大负载即缺省并行计算的最大作业量，updateTime代表该节点资源状态更新的时间戳。status代表节点的在线状态，在cn.edu.hrbeu.theweb.logic.DEF类里设置了三个枚举值；loads表示当前节点负载即正在运行的作业数量。solvers字段是一个json数组，为该计算节点已经安装的所有求解器的_id集合，以下是计算节点status字段属性。

表4.9: 计算节点status状态图

状态 含义

_AGENT_SYN_STATUS_UNKNOW = -1; 未知状态

_AGENT_SYN_STATUS_OFF_LINE = 0 离线状态

_AGENT_SYN_STATUS_ON_LINE = 1 在线状态

可通过查询数据库的方式获取收集各agent的资源状态信息，主要代码如下：、

表4.10 收集agent资源状态

Document agentItem = DB.agent.__get_by_id(this.agentId, DEF._IP,

DEF._PORT, DEF._SOLVERS,

DEF._STATUS, DEF._LOADS);

4.7 本章小结：{ 62% : 本章论述了节点管理模块的功能， }重点讲述了控制节点leader与计算节点agent互相

通信时维护的心跳机制，leader给agent发送消息的三种消息格式“syn”，“taskRun”，“taskKill”。以及leader接收agent消息响应后的消息反馈和被动格式收集集群资源状态数据。

第5章 集群任务分发与负载均衡调度策略的实现5.1需求分析当服务器停留在单机架构时，{96%：随着业务的增加，访问量的提高和数据的不断堆积，系统会变得越来越慢，硬件设备的更新换代是一个必不可少的工作。 }{99%：在达到同等性能的条件下，采用计算机集群比采用同等能力的计算机所花的代价要小很多， }能降低成本；{93%：采用传统服务器的用户如果需要大幅度扩展系统的能力，就必须购买昂贵的服务器， }{ 70%：将单机架构过渡到集群技术，只需要将新的服务器加入集群即可， }相比而言提高了系统的扩展性。作业调度平台中心控制系统相当于一个代理负载均衡器，整个系统则为负载均衡集群。{82%：每个计算节点都可以承担一定的处理负载，并且可以由中心控制节点实现处理负载在计算节点之间的动态分配。 }{ 66%：作业调度平台中心控制节点应用服务要求分布式集群检查每个计算节点检查当前的负载，并确定哪些节点可以接受新的作业， }以实现集群的任务分发。

5.2总体实现中心控制节点服务在LeaderServer.init()函数中进行启动，首先启动LeaderTaskSender线程实例处理任务分发，接着获取agent数据表中所有计算节点的集合agentList。遍历该agentList, 中心控制节点启动相应数量的NextAgentKeeper线程对下级计算节点进行任务状态处理以及消息通信，关键性代码如下：

表5.1 LeaderServer.init()函数原型

```
public static void init() {
// 启动任务发射
LeaderTasksSender.getInstance();
// 启动相应数量的NextAgentKeeper线程
List<Document> list = DB.agent.__list(null, null, Docat.__select(DEF._ID));
for (Document d : list) {
DB.agent.__update_one(d,
new Document(DEF._STATUS, DEF._AGENT_SYN_STATUS_UNKNOW),
null, -1,
Docat.__select(DEF._ID));
NextAgentKeeper.get(Docat.getString(d, DEF._ID));
}
}
```

LeaderServiceResource, LeaderService， LeaderTaskSender, NextAgentKeeper四个类的交互时序图如下

图5.1: 任务分发时序图

LeaderFeedbackResource是Leader对Agent的内部反馈接口。当请求路径为/leader可以直接跳转到agentApi()函数，用于处理对agent的api请求。agentApi()会调用LeaderFeedbackService.router(), 处对agent的具体请求，如“solverSyn”求解器同步，“taskSyn”任务同步。然后由router()路由函数跳转到具体处理该请求的函数。当请求路径为/ leader/solver/file时，会跳转到agentSolverFileDownload函数用于处理对agent的文件同步。

5.3 集群任务分发LeaderTaskSender类是专门用来处理集群任务分发，继承自java.lang.Thread类，实现方式使用了单例设计模式，LeaderTaskSender类只有一个实例，{ 57%：该类由getInstance方法自行创建该实例， }向整个系统公开这个实例接口。

LeaderTaskSender的启动在创建该对象实例instance时进行。

表5.2 LeaderTaskSender的实例启动

```
public static LeaderTasksSender getInstance() {
if (instance == null) {
instance = new LeaderTasksSender();
instance.start();
```

```
}
return instance;
```

{100% : Thread的run方法是不抛出任何检查型异常(checked exception)的,但是它自身却可能因为一个异常而被终止，导致这个线程的终结。 }{100% : 最麻烦的是，在线程中抛出的异常即使使用try. }{100% : ..catch也无法截获，因此可能导致一些问题出现，比如异常的时候无法回收一些系统资源，或者没有关闭当前的连接等等。 } LeaderTaskSender在初始化静态块中为所有线程加入了异常处理，{ 60% : 设置了线程异常的打印和处理信息。 }

表5.3: LeaderTaskSender类异常处理

```
static {
Thread.setDefaultUncaughtExceptionHandler(new Thread.UncaughtExceptionHandler() {
@Override
public void uncaughtException(Thread t, Throwable e) {
if (t instanceof LeaderTasksSender) {
System.err.println(new SimpleDateFormat("HH:mm:ss_SSS> ").format(new Date()) +
"LoggerThread " + ((LeaderTasksSender) t).getThreadName() +
" died! Because " + E.formatThrowable(e));
System.exit(9);
} else {
System.err.println(new SimpleDateFormat("HH:mm:ss_SSS> ").format(new Date()) +
"Thread " + t.getName() + " died! Because " + E.formatThrowable(e));
}
}
});
}
```

在进行集群任务分发时，必要的基础条件时获取当前有效的计算节点资源状态。getValidAgentList方法首先构建求解器映射表，求解器映射表是一个Map<String, Document>数据结构。

其次获取活跃的计算节点列表，接着使用DB._list(Document where, Document order, Document select)非分页方式获取获取所有在线计算列表，此处的参数where是查询条件，值为new Document(DEF._STATUS, DEF._AGENT_SYN_STATUS_ON_LINE), 即要查询的条件为{“status” : 1}的所有计算节点，参数order值为NULL表示不排序，select的参数值表示查询结果投影的字段，此处为DEF._LOADS, DEF._MAX_LOADS，DEF._SOLVERS， DEF._PATH。将这些查询信息添加到agenList里。

接着遍历该agentList, 首先获取agentID, loads, maxLoads字段，比较计算节点当前负载量loads和最大设置负载量maxLoads。如果当前作业负载量小于最大负载量，首先获取该计算节点的所有求解器ID列表solverIds, 判断该agent是否配置了solver, 如果配置了solver则从求解器映射表里获取该solver的相关属性solverItem, 并将其添加到solverList里。如果solverList不为空，则将其添加到activeAgentlist中。如果activeAgentList中活跃的计算节点数大于2，则根据每个节点的剩余负载量（最大负载量-当前负载量）对activeAgenList活跃计算节点列表进行一个排序，返回活跃计算节点列表剩余负载量的一个升序版本。

在DEF类中对Task求解任务的每一个状态进行标识。

表5.4: Task属性标识

状态 含义

_TASK_RUN_STATUS_CREATE = 0 未启动

_TASK_RUN_STATUS_PENDING = 1 等待启动

_TASK_RUN_STATUS_HAS_= 2 已启动

_TASK_RUN_STATUS_IS_RUNNING = 3 执行中

_TASK_RUN_STATUS_IS_KILLED = 4 强制停止

_TASK_RUN_STATUS_HAS_FINISHED = 5 执行结束

_TASK_RUN_STATUS_RECORD_LOST = -1 任务不存在

_TASK_RUN_STATUS_FAILED_STARTED = -2 启动失败

_TASK_RUN_STATUS_EXCEPTION_STOP = -3 意外停止

_TASK_RUN_STATUS_IS_CANCLED = -4 取消

_TASK_RUN_STATUS_SOLVER_NOT_EXIST = -5 求解器不存在

_TASK_RUN_STATUS_SOLVER_NOT_DIST = -6 求解器未发布
_TASK_RUN_STATUS_AGENT_INTERNAL_ERROR = -7 计算节点内部错误(重启)

_TASK_RUN_STATUS_TASK_CANNOT_CREATE = -8 任务无法创建

doSend()方法是处理集群任务分发的函数，任务分发调度策略策略展示流程图如下：

图 5.2: LeaderTaskSender类图

5.4 负载均衡调度策略实现5.4.1获取未分配Task列表由getUndistTaskList()方法先获取未分配的task列表存储在taskList中。并设置计数器count,初始值设置为0,代表将要分发的任务个数。

表5.5 获取未分配的Tak列表

```
List<Document> taskList = this.getUndistTaskList();
if (taskList == null || taskList.isEmpty()) {
return 0;
}
int count = 0;
```

getUndistTaskList()方法原型如下：首先设置查询条件为undistTaskWhere, 即以任务状态为_TASK_RUN_STATUS_PENDING = 1 等待启动的任务为过滤条件。排序方式为按照_ID降序。在获取未分配Task列表时对线程对象进行加锁，保证数据的可靠性。

表5.6: getUndiskTaskList方法原型

```
private final Document undistTaskWhere = new Document(DEF._STATUS,
DEF._TASK_RUN_STATUS_PENDING)
.append(DEF._VERSION, DEF._NOT_EXISTS)
.append(DEF._AGENT, DEF._NOT_EXISTS);
private final Document undistTaskOrder = Docat.__order_down(DEF._ID);
private final Document undistTaskSelect = null;
private List<Document> undistTaskList;
private boolean reload = true;
private List<Document> getUndistTaskList() {
synchronized (this) {
if (this.reload) {
this.reload = false;
undistTaskList = DB.task.__list(undistTaskWhere, // 可分配但未分配
undistTaskOrder, undistTaskSelect);
}
return undistTaskList;
}
}
```

5.4.2构建求解器映射表首先根据undistTaskList未分配Task列表，获取所有相关task对应求解器Solver的编号_ID.存储在以List<string>为数据结构的solverIds变量中。其次以DEF._VERSION, DEF._LOADER, DEF._EXECUTOR字段构建求解器映射表solverMap。

表5.7 构建求解器映射表

```
List<String> solverIds = Docat.__get_value_list(undistTaskList, DEF._SOLVER, true);
Map<String, Document> solverMap = DB.solver.__map(new Document(DEF._ID, DEF._IN(solverIds)),
null,
Docat.__select(DEF._VERSION,
DEF._LOADER,
DEF._EXECUTOR));
```

5.4.3填充求解器映射表找到每一个求解器可以分配的节点列表。

首先先找到solver在每个agent上的分配情况。项目中的dist数据库表是专门用来连接solver和agent的。比如名称为E209，_ID为5c921bd7f589d423d0df9046的计算节点可以映射一个或多个求解器。dist数据库表的结构如下：

表5.8: dist数据表结构

```
{
"_id" : "5c921bd7f589d423d0df9046",
"agent" : "5c90d3c7f589d41918717c08",
"solver" : "59fee826f589d42b7864def2",
"version" : 4,
"status" : 3,
"createTime" : NumberLong(1553079255314)
}
```

Status为3,表示_ID为59fee826f589d42b7864def2的solver求解器在该节点已经同步成功。

获取初级比较指标loads。在distinct数据表中获取agentID字段的不重复值：List agentIds = DB.dist.__distinct(DEF._AGENT, where, null);如果agentIds不为空，则从agent表中获取以DEF._LOADS, DEF.MAX_LOADS, DEF.PATH为集合的agent列表agentList。如果agentList不为空，则遍历agentList。从agentList取出每一个成员项agent,获取相应的agentID, loads, maxLoads。

表5.9 获取初级比较指标loads

```
if (agentIds != null && !agentIds.isEmpty()) {
List<Document> agentList = DB.agent.__list(
new Document(DEF._ID, DEF._IN(agentIds)).
append(DEF._SUSPEND, false),
null,
Docat.__select(DEF._LOADS, DEF._MAX_LOADS, DEF._PATH));
if (agentList != null && !agentList.isEmpty()) {
for (int j = agentList.size() - 1; j >= 0; j--) {
Document agentItem = agentList.get(j);
String agentId = Docat.getString(agentItem, DEF._ID);
int loads = Docat.getInteger(agentItem, DEF._LOADS, 0);
int maxLoads = Docat.getInteger(agentItem, DEF._MAX_LOADS, 0);
if (loads >= maxLoads) {
```

```
agentList.remove(j);
}
}
```

获取准确比较指标loads。如果当前遍历agent节点的作业负载loads大于最大负载阈值maxLoads,则将该agent从agentList中移除。否则获取准确负载后再比较一次，从Task数据表中统计该agent的所有作业负载，包括正在运行，等待启动和已经启动的任务。如果获取的准确负载loads大于最大负载阈值maxloads,则从agentList中移除该节点。否则，计算该节点剩余负载，即leftloads = maxLoads – loads。{ 59% : 并将该字段项写入agent数据表中。 }

表5.10 获取准确比较指标loads

```
else {
// loads只是初级指标，获取准确负载后再比较一次
loads = DB.task.__count(new Document(DEF._AGENT, agentId).
append(DEF._STATUS, DEF._IN(DEF._TASK_RUN_STATUS_IS_RUNNING,
DEF._TASK_RUN_STATUS_PENDING,
DEF._TASK_RUN_STATUS_HAS_STARTED)));
if (loads >= maxLoads) {
agentList.remove(j);
} else {
int leftLoads = maxLoads - loads;
canLoads += leftLoads;
agentItem.append(DEF._LEFT_LOADS, leftLoads);
}
}
```

将可用的agent列表写进solver表字段中，并且将可用的agent按照leftLoads降序排列，目的是将任务分配到最空闲的agent中。

表5.11 对agent按照leftLoads降序排列

```
if (size > 0) {
// 将可用的agent放到solver中
solver.append(DEF._AGENTS, agentList);
// 如果需要按照leftLoads降序排列，目的是将任务分配到最空闲的agent中
this.reorderAgentList(agentList);
}
```

reorderAgentList是对agent按照leftLoads剩余负载进行排序的函数，其原型如下

表5.12 reorderAgentList函数原型

```
private void reorderAgentList(List<Document> agentList) {
if (agentList.size() >= 2) {
agentList.sort(new Comparator() {
@Override
public int compare(Object o1, Object o2) {
int l1 = Docat.getInteger((Document) o1, DEF._LEFT_LOADS);
int l2 = Docat.getInteger((Document) o2, DEF._LEFT_LOADS);
if (l1 > l2) {
```

```
return -1;
} else if (l1 == l2) {
return 0;
} else {
return 1;
}
}
});
}
}
```

5.4.4将Task按顺序进行分配获取等待启动的任务Task：从task数据表中获取任务状态为等待启动的任务即标识为_TASK_RUN_STATUS_PENDING则获取该taskIntem。如果taskItem为空，则该task在分配前已经被取消，打印日志信息并且从taskList中移除该task。

表5.13 获取等待启动的任务Task

```
for (int i = taskList.size() - 1; i >= 0; i--) {
Document taskItem = DB.task.__get(
New Document(DEF._STATUS, DEF._TASK_RUN_STATUS_PENDING).
append(DEF._ID, taskList.get(i).get(DEF._ID)),
null,
null
);
if (taskItem == null) {
M.trace("TASK分配前已被取消 => ", Docat.__doc_to_json(taskList.get(i)));
taskList.remove(i);
continue;
}
```

获取该task关联的求解器映射信息：从task数据表中的每一项taskItem取得taskId, solverId。然后根据该solverId从求解器映射表solverMap中获取该solver的映射信息solverItem。如果solverItem为空，则更新task表status字段为求解器不存在即_TASK_RUN_STATUS_SOLVER_NOT_EXIST，并追加日志信息字DEF._MESSAGE为“求解器已经删除”，添加相应的时间戳。

表5.14 获取该task关联的求解器映射信息

```
String taskId = Docat.getString(taskItem, DEF._ID);
String solverId = Docat.getString(taskItem, DEF._SOLVER);
Document solverItem = solverMap.get(solverId);
if (solverItem == null) {
DB.task.__update_by_id(taskId,
new Document(DEF._STATUS,
DEF._TASK_RUN_STATUS_SOLVER_NOT_EXIST)
.append(DEF._MESSAGE, "求解器被删除"),
System.currentTimeMillis(), DEF._ID);
taskList.remove(i);
```

}

任务分配与后处理：如果求解器不为空，首先从该solver求解器映射表中获取该求解器可分配的计算节点列表agentList。如果agentList不为空则逐项进行任务分配，否则留待以后处理。遍历agentList的每一项，获取agentId以及剩余负载leftLoads, 如果leftLoads大于0，则中心控制节点将发送任务运行消息到该计算节点启动求解任务计算，如果任务启动成功，则从任务列表taskList中移除该任务，更新agent表中leftLoads为leftLods – 1，最后对agentList中的所有节点进行重新排序.计算器count增1，doSend（ ）函数返回成功分配的任务个数。

表5.15 任务分配与后处理

```
else {
List<Document> agentList = Docat.getList(solverItem, DEF._AGENTS);
if (agentList != null && !agentList.isEmpty()) {
// 不空则逐项分配，否则留待以后处理
for (Document agentItem: agentList) {
String agentId = Docat.getString(agentItem, DEF._ID);
int leftLoads = Docat.getInteger(agentItem, DEF._LEFT_LOADS, 0);
if (leftLoads > 0) {
int c = NextAgentKeeper.get(agentId).
sendTaskRunMessage(taskItem, solverItem, agentItem);
if (c > 0) {
taskList.remove(i);
agentItem.append(DEF._LEFT_LOADS, leftLoads - 1);
this.reorderAgentList(agentList);
count++;
break;
}
}
}
}
}
return count;
```

5.5 节点任务状态处理NextAgentKeeper类是中心控制节点用于保持对下级节点的任务状态处理的类，也继承自java.lang.Thread类，其私有成员里有以Map<String, NextAgentKeeper>为数据结构的_map。Key为_id，Value为NextAgentKeeper实例。 agentId表示计算节点id, errCount表示计算节点上运行错误的任务量计数器。lastTime表示结束时间戳，isKilled为布尔变量表示中心控制节点对该下级节点管理的线程是否结束。

queue是实现了LinkedBlockingQueue<Document>数据结构的阻塞消息队列，{88% : LinkedBlockingQueue是一个线程安全的阻塞队列，实现了先进先出等特性，是作为生产者消费者的首选，可以指定容量，也可以不指定，不指定的话默认最大是Integer. }MAX_VALUE，{100% : 其中主要用到put和take方法，put方法将一个对象放到队列尾部，在队列满的时候会阻塞直到有队列成员被消费，take方法从head取一个对象，在队列为空的时候会阻塞，直到有队列成员被放进来。 }构造方法将每个线程实例绑定一个agentId。

表5.16 NextAgentKeeper构造方法

```
public NextAgentKeeper(String agentId) {
this.agentId = agentId;
```

}

5.5.1 成功状态处理onSucceed(int status)成功状态处理函数，更新时间戳，设置错误计数器errCount为0，更新该线程实例相关联的计算节点agent状态status为在线状态。

表5.17 onSucceed函数原型

```
private void onSucceed(int status) {
this.lastTime = System.currentTimeMillis();
this.errCount = 0;
this.setAgentStatus(status, DEF._AGENT_SYN_STATUS_ON_LINE);
}
```

5.5.2 未知错误处理onFault(int status)未知错误处理函数，更新时间戳，并更新该线程实例相关联的计算节点agent状态status为_AGENT_SYN_STATUS_UNKNOW未知状态。

表5.18 onFault函数原型

```
private void onFault(int status) { // 不该出错的
this.lastTime = System.currentTimeMillis();
this.setAgentStatus(status, DEF._AGENT_SYN_STATUS_UNKNOW);
}
```

5.5.3 错误处理onError(int status)错误处理函数，更新时间戳，更新错误处理计数器errCount增1，如果errCount大于3，则设置该线程实例相关联的计算节点agent状态status为_AGENT_SYN_STATUS_OFF_LINE离线状态。

表5.19 onError函数原型

```
private void onError(int status) {
this.lastTime = System.currentTimeMillis();
this.errCount++;
if (this.errCount >= 3) {
this.setAgentStatus(status, DEF._AGENT_SYN_STATUS_OFF_LINE);
}
}
```

5.5.4 任务分配失败处理onTaskRunSendErr(Document task)任务分配失败处理函数，将以该task为参数的任务其中一些字段进行重置。

表5.20 onTaskRunSendErr函数原型

```
private void onTaskRunSendErr(Document task) {
String taskId = Docat.getString(task, DEF._ID);
Document value = new Document();
value.append(DEF._AGENT, 0);
value.append(DEF._VERSION, 0);
value.append(DEF._DIST_TIME, 0);
DB.task.__unset_one(new Document(DEF._ID, taskId), value, -1, null);
}
```

5.6 节点任务消息处理NextAgentKeeper中的路由函数为handleMessage(Document message)。当message中action字段为“syn”时，调用doSynSend()方法同步求解器和状态信息；action字段为“taskRun”时，调用doTaskRunSend()方法，中心控制节点方法发送任务启动命令到相应线程实例关联的计算节点上。action字段为“kill”时, 调用doTaskKillSend()方法,中心控制节点方法发送任务结束命令到相应线程实例关联的计算节点上。否则，打印消息NextHostKeeper 不支持下发消息类型。

表5.21 handleMessage路由函数原型

```
private void handleMessage(Document message) {
String action = Docat.getString(message, DEF._ACTION, "syn");
if ("syn".equals(action)) {
M.trace("NextHostKeeper handleMessage => ",
this.agentId, " ", Docat.__doc_to_json(message));
this.doSynSend();
} else if ("taskRun".equals(action)) {
M.trace("NextHostKeeper handleMessage => ",
this.agentId, " ",
Docat.__doc_to_json(message));
this.doTaskRunSend(Docat.getDocument(message, DEF._TASK));
} else if ("taskKill".equals(action)) {
M.trace("NextHostKeeper handleMessage => ",
this.agentId, " ",
Docat.__doc_to_json(message));
this.doTaskKillSend(message);
} else {
M.trace("NextHostKeeper 不支持下发消息类型 => ",
this.agentId, " ",
Docat.__doc_to_json(message));
}
}
```

sendSolverSynMessage(String solverId)函数发送求解器同步消息。同步以solverId为编号的求解器。

表5.22 sendSolverSynMessage函数原型

```
public void sendSolverSynMessage(String solverId) {
Document message = new Document(DEF._ACTION, "syn").
append(DEF._SOLVER, solverId);
this.sendMessage(message);
}
```

sendTaskKillMessage(Document taskItem)函数发送结束任务消息，参数为taskItem。

表5.23 sendTaskKillMessage函数原型

```
public void sendTaskKillMessage(Document taskItem) {
Document message = new Document(DEF._ACTION, "taskKill").append(DEF._ID,
Docat.getString(taskItem, DEF._ID));
this.sendMessage(message);
}
```

sendMessage与sendSolverMessage和sendTaskKillMessage方法的调用过程如下：

图5.3 sendMessage函数的引用关系

sendMessage(Document message)传递Document为类型的message，对阻塞队列queue进行加锁。将该消息message加入队列并唤醒所有线程并打印日志信息。

sendTaskRunMessage(Document taskItem, Document solverItem, Document agentItem)函数有以Document为类型的三个参数taskItem， solverItem， agentItem。首先根据taskItem获取taskId, 接着根据solverItem获取求解器版本号version。设置查询条件where，以及更新值update对task表记录进行更新。添加”taskRun”命令到阻塞队列queue中，并唤醒其它所有线程。

表5.24 sendTaskRunMessage函数原型

```
public int sendTaskRunMessage(Document taskItem, Document solverItem, Document agentItem) {
String taskId = Docat.getString(taskItem, DEF._ID);
int version = Docat.getInteger(solverItem, DEF._VERSION);
Document where = new Document(DEF._ID, taskId)
.append(DEF._STATUS, DEF._TASK_RUN_STATUS_PENDING)
.append(DEF._VERSION, DEF._NOT_EXISTS)
.append(DEF._AGENT, DEF._NOT_EXISTS);
Document update = new Document(DEF._VERSION, version)
.append(DEF._AGENT, agentId)
.append(DEF._DIST_TIME, System.currentTimeMillis())
.append(DEF._PATH, Docat.getString(agentItem, DEF._PATH))
.append(DEF._LOADER, Docat.getString(solverItem, DEF._LOADER))
.append(DEF._EXECUTOR, Docat.getString(solverItem, DEF._EXECUTOR));
synchronized (queue) {
Document task = DB.task.__update_one(where, update, null,
System.currentTimeMillis(),
null);
if (task != null) {
Document message = new Document(DEF._ACTION, "taskRun").
append(DEF._TASK, task);
this.queue.add(message);
this.queue.notifyAll();
return 1;
}
}
return 0;
}
```

setAgentStatus(int oldStatus, int newStatus)方法设置agent节点的状态。

表5.25 setAgentStatus函数原型

```
private void setAgentStatus(int oldStatus, int newStatus) {
if (oldStatus != newStatus) {
DB.agent.__update_by_id(this.agentId,
new Document(DEF._STATUS, newStatus),
-1, DEF._ID);
}
}
```

NextAgentKeeper线程实例在get(String _id)方法中启动线程。

表5.26 NextAgentKeeper线程实例启动

```
public static NextAgentKeeper get(String _id) {
synchronized (_map) {
NextAgentKeeper ak = _map.get(_id);
if (ak == null) {
M.trace("启动 NextHostKeeper => ", _id);
ak = new NextAgentKeeper(_id);
ak.start();
_map.put(_id, ak);
}
return ak;
}
}
```

get(String _id)方法相关引用图如下：

图5.4 get函数的引用关系

5.7 本章小结本章是关于作业调度平台中心控制节点对集群进行任务分发，中心控制节点对下级计算节点进行管理通信，以及调度策略的设计与实现。集群任务分发和对下级节点进行管理的核心是对线程，数据库表task, dist, solver, agent数据进行查询更新与删除操作，以及多种边界异常的处理。调度策略的设计与实现主要依据求解器映射表的填充，每个solver类型的求解任务会映射一个或多个按照leftLoads进行降序排列的agent，当发送具体任务时，可以根据求解器映射表将该求解器相关任务优先分配到剩余负载最大的计算节点上，并更新相应task, agent, 求解器映射表等有关状态字段，为下一次求解任务分发做准备。

第6章 系统展示与分析本章介绍系统前端界面展示与后台逻辑的关联分析，在详细分析的基础上，{ 56% : 对各个功能模块进行测试联调，完成整个系统的集成， }对求解器按需同步机制与调度策略算法进行验证。

6.1 总体设计（1）求解器按需部署与自动同步机制

（2）任务分发与调度策略

结 论作业调度平台中心控制系统的核心是求解器管理模块，节点管理模块，集群任务分发和调度策略的实现。

(1) 理论价值：{ 69% : 随着中国数值水池网站虚拟实试验的不断集成， }爆炸式海量数据处理逐渐成为单机服务器的重要瓶颈，通过增加硬件方式提升数据处理能力的传统方式存在投资资本大的缺点，{ 65% : 分布式集群技术通过特定连接方式， }将廉价服务器结合起来，从而提高作业平台的并行求解与处理能力。系统提出一种基于代理服务器的模型，将求解器管理，节点管理，任务分发和调度策略等功能通过该代理服务器即中心控制节点来执行，从而使得集群整体性能最优。

（2）实用价值： 求解器管理模块的功能实现解决了系统维护人员在分布式集群环境下手工迭代各计算节点求解器版本文件的难题，大大提高了工作效率。集群任务分发使得所有求解任务在分布式环境中运行有条不紊，调度策略能综合考量集群各计算节点资源利用率，求解器映射表以最高效的方式将等待启动的求解任务从队列中依次出队与计算节点进行映射。将求解器列表，计算任务列表，计算节点列表等模块可视化展示在浏览器界面上，将整个集群所有资源状态集中展示，方便系统管理员对整个系统进行扩展和性能评估。

（3）遗留问题及改进方案： 中心控制节点仅仅用集群各计算节点当前的作业数量反映计算节点的真实负载情况，还需要对系统的性能，相应时间，I/O和网络带宽等参数进行剖析，可以采用双机热备份负载均衡器，同时引入一个负载冗余以动态调整节点负载分配，从而达到尽量简化中心控制节点的任务分配算法、最大限度满足系统最大吞吐率和提高系统响应时间的目标，真正做到将计算集群的求解效率提高；

s

参考文献[1] Karatza H D . A simulation model of task cluster scheduling in distributed systems[C]// IEEE Workshop on Distributed Computing Systems. IEEE, 1999.

[2] Neamatollahi P , Naghibzadeh M , Abrishami S , et al. Distributed Clustering-Task Scheduling for Wireless Sensor Networks Using Dynamic Hyper Round Policy[J].{ 73% : IEEE Transactions on Mobile Computing, } 2018.

[3] Zhang B Y , Mo Z Y , Yang G W , et al.{ 69% : Scheduling Efficiently for Irregular Load Distributions in a Large-scale Cluster[C]// International Conference on Parallel & }{ 62% : Distributed Processing & Applications. }{ 71% : Springer-Verlag, 2005. }

[4] Kanemitsu H , Lee G , Nakazato H , et al. Static task cluster size determination in homogeneous distributed systems[M]// Software Automatic Tuning. Springer New York, 2011.

[5] Chen W , Rafael F D S , Deelman E , et al. Using imbalance metrics to optimize task clustering in scientific workflow executions[J]. Future Generation Computer Systems, 2015, 46:69-84.

[6]汪洋. Linux集群数据生成及管理工具中任务管理模块的设计与实现[D]. 2014.

[7]杨义彬. 基于云计算的分布式处理框架的研究与设计[D]. 电子科技大学, 2011.

[8]李博, 袁曙涛, 魏晓辉, et al. 一个综合性集群监测模型MCM的设计与实现[J]. 吉林大学学报(理学版), 2008, 46(01).

[9]燕明磊. Hadoop集群中作业调度研究[J]. 软件导刊, 2015(4):1-2.

[10]杨冬菊, 胡正国. 应用代理机制实现异构集群系统的管理[J]. 西北工业大学学报, 2001(4):644-647.

[11]邓景文. 集群系统下面向用户的作业公平调度算法[D]. 北京邮电大学, 2008.

{86% : [12]安喜锋. 高性能计算集群管理系统与作业调度技术研究与实现[D]. 西北工业大学, 2005. }

[13]王越峰. 基于Hadoop集群的作业调度算法研究与改进[D].

[14]杨冬菊. 异构集群管理系统的关键技术研究[D]. 西北工业大学, 2002.

[15]董世龙. 基于模糊聚类的云任务调度优化策略研究[D]. 广西大学, 2014.

[16]田珍. 基于集群技术的作业管理系统研究与应用[D]. 西北工业大学, 2006.

[17]魏士祥. 面向过程感知的云作业资源调度[D]. 南京理工大学, 2014.

[18]何林. 面向网格计算的多集群间作业调度策略的设计与实现[D]. 哈尔滨工业大学.

[19]刘海龙. 基于集群技术的作业管理系统[D]. 西北工业大学, 2005.

[20]喻俊, 艾迪. 计算机网络集群分布式调度方法[J]. 金卡工程, 2013(10):25-26.

[21]丁晶晶. MapReduce框架下的任务调度算法研究[D]. 2017.

[22]唐一韬, 黄晶, 肖球. 一种基于DAG的MapReduce任务调度算法[J]. 计算机科学, 2014, 41(s1):42-46.

[23]黄展智. 物联网任务分布式调度策略的研究[D]. 2016.

[24]应对平台高并发的分布式调度框架TBSchedule[J]. 电脑编程技巧与维护, 2016(7):4-4.

[25]乔兵, 孙志俊, 朱剑英. 基于Agent的分布式动态作业车间调度[J]. 信息与控制, 2001, 30(4):292-296.

致谢四年前，{ 63% : 我怀着对象牙塔的憧憬和神圣殿堂的向往，带着纷繁的心情和梦想，第一次来到了哈尔滨这座城市，一切都是新奇与陌生，从最初高考后懵懵懂懂的选择了软件工程专业， }到如今四年的时光已经悄然逝去，我很清楚的明白自己当初的选择是正确的，人生中坚持并一直着手去做自己喜欢并感兴趣的东西是很难得的。{ 64% : 大三那一年，除了学好本专业课程外，把剩余的课余时间全都花在了导师指导的科研项目和实习工作中， }在导师的指导中以及实习工作的磨砺中，{ 76% : 我感觉到了自己的不足，也感受到了自己一天天成熟带来的变化。 }

感谢您，我的导师，{ 76% : 我的大学，正是在给予我的每一种挑战与机遇中，让我懂得了学习与生活。 } { 67% : 懂得时间与成长，懂得奋斗和拼搏。 }{86% : 是你为我架起了走向成熟的桥梁，在慢慢人生的长路上，你将是我多彩世界永远美丽与难忘的记忆。 }{94% : 品味人生，才能珍惜人生。走过岁月，走过季节，也走过我的大学时代。 }{100% : 大学生活留给我更多的时间去思考。 }{100% : 想想走过的路，想想现在的路，想想未来的路，不能不说是对自己的重新认识。 }

{94% : 六月了，我已经闻到了离别的气息，四年的大学生活也将结束，我也终于明白“转瞬即逝”的含义。 }{98% : 大四，在这即将远离大学时代的时刻，才真正懂得回眸的意义， }再见青春，永恒的迷惘。

检测报告由PaperTime文献相似度检测系统生成