为了尽可能的保证数据的安全性, 将后台管理与前台交易分别部署在内网区 (SA) 和互联网区 (IA). 互联网区域主要提供交易服务, 交易实时数据存放在互联网区的交易数据库; 内网区主要提供后台管理及控制服务, 管理数据存放到后台管理数据库.

在向云平台迁移之前, SA 和 IA 区是通过防火墙实现逻辑隔离, 前后台可以通过映射方式访问. 云平台使用“安全隔离网闸”进行隔离, 前后台区域的数据链路不是实时连通的, SA-IA 数据交换模式上采用半连接式的文件交换形式, 这种方式同一时刻只允许一个方向的访问. 这种数据交换方式, 在性能方面显然是存在短板的, 因为需要牺牲实时性, 而交易过程中的一些管理控制信息, 是需要实时传送到交易服务器上的.

为在保证安全的前提下实现数据的高速交换, 本文把交易系统的数据流进行了进一步的拆分, 划分交易数据、控制数据、管理数据 3 部分. 交易数据的内容和迁移前相似; 而迁移前的后台管理数据拆分为控制数据和管理数据. 管理数据直接进入管理后台, 而控制数据通过开发新的控制程序直接进入到交易前端. SA-IA 的数据交换依托于文件交换结合 SA-IA 数据交换模块进行, 云平台环境下本系统结构见图 3.

图 3 数据分流设计

传统交易系统由资源监控程序、资源池、线程监控程序、线程池、同步监控程序、同步程序、消息池等 7 部分组成[11], 见图 4.

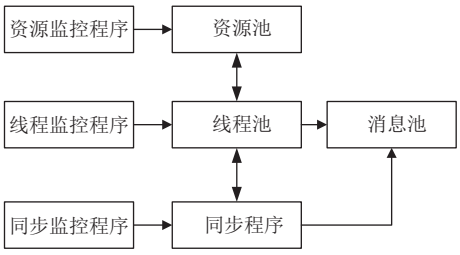

图 4 传统交易系统内核结构

云服务器依托云平台的虚拟化技术, 对外表现出很高的稳定性, 其通过虚拟化管理策略能轻松的实现故障转移, 然而正式这种自动化的故障转移机制, 让交易系统显得有些水土不服. 主要是因为交易系统有自身的交易引擎, 这个交易引擎是基于数据的池化技术实现的. 云服务器自动故障转移后, 交易系统引擎中的实时数据并不能实时转移到新的云服务器中. 因此在交易系统中增加交易接管机制来适应云平台的这种故障转移, 一方面当云服务器进行自动故障转移时, 将交易引擎的数据进行同步转移, 另一方面在引擎运行期间对交易系统中的状态进行监控, 对各资源池、消息池、线程池等进行清理净化, 见图 5.

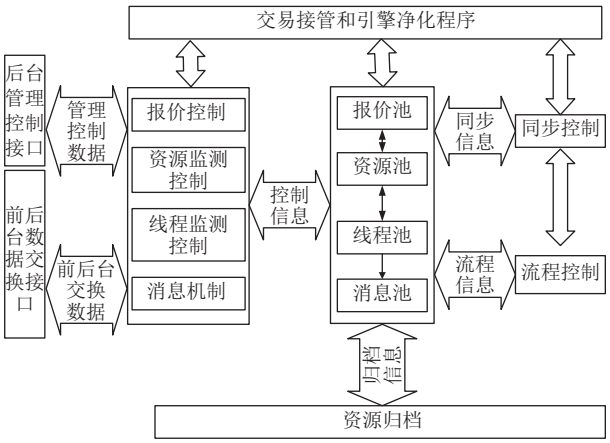

图 5 云平台下的交易系统内核引擎结构

同时为提高系统性能和稳定性, 引入资源归档程序、同步控制等模块, 这些模块设计和算法相对简单, 文中就不在赘述.

3 主要算法

本节分别介绍处理流程中一些重要过程的主要内容, 并给出具体算法的主要步骤.

3.1 SA-IA 数据交换算法

SA-IA 数据交换采用数据“不落地”方式进行, 即将需要交换的数据以哈希表的方式放入交换子系统缓存, 需要交换时直接从缓存读取并进行交换, 交换模块简称 DESS.

DESS 模块分为: 前台交换接口 DE_SI、 前台交换模块 DE_SM、 内网交换接口 DE_II、内网交换模块 DE_IM 四部分.

DE_SM 模块主要功能包括: 1) 接收前台交易系统

发来需要交换的数据 DE, 并放入哈希表 HT_0101, 供 DE_IM 读取; 2) 将内网区发来的需要交换的数据接收到 HT0102, 并进行业务加工处理. 过程见算法 1.

DE_IM 模块主要功能: 一是接收后台管理系统发来的数据并放入哈希表 HT_0201, 供 DE_SM 读取; 将后台发来的需要交换的数据接收到 HT0202 并进行业务加工处理.

为保证数据交换的同步性和性能, 每条交换数据 DE 设置一个状态字 S 和一个唯一值 K, DE={K, S, D}, 状态 S 记录数据交换结果, D 是需要交换的数据.

DE_IM 模块的过程同 DE_SM 类似, 文中就不再赘述.

算法 1. DE_SM 数据交换算法

```
Input: DE
Output: HT0101, HT0102
While DE in SA BUFFER
  add DE to HT0101
End while
HT0102 ← HT0201//accept HT0201 from Intranet to HT0102
Foreach DE in HT0102
  Dealing DE
End foreach
Return TH_0101 HT_0102
```

3.2 交易接管和引擎净化

云平台的故障转移、接管, 为系统提供了更高的可用性. 交易系统采用负载均衡集群模式部署, 交易业务被按一定算法部署在不同的服务器上, 而相同业务的交易由至少 2 台服务器以互为负载均衡的方式承担, 如图 6 所示. 在图中, A 为 B 的负载服务器, B 为 A 的负载服务器, 并提供一定数量的备用机. 以两台服务器为例, 两台服务器的 HTTP 请求的会话, 交易内核数据是实时同步的, 只要有其中一台正常运行, 系统就能正常使用. 但是, 当负载中的一台服务器出现故障后, 采用一台服务器提供交易只能作为过渡策略, 必须研究可行的交易接管方法, 保证系统的稳定性和及时响应. 主要过程包括: 实时负载检测、交易同步、自动交易接管、交易引擎自动净化等. 通过增加这些处理过程, 实现故障交易主机自动接管, 并提高了交易引擎的性能和稳定性.

负载检测模块: 查询交易主机配置信息表 BASIT, 获取与之对应的负载主机信息. 主服务器发送同步检查消息, 查看负载主机是否正常运行, 如果正常运行, 标记为可用状态, 如果未正常运行, 标记为不可用状态.

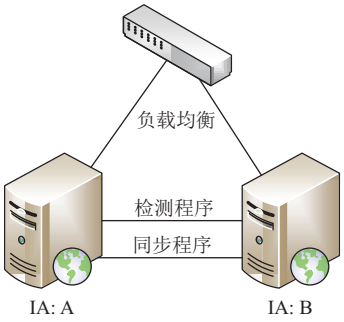

图 6 负载交易模式

优化后的同步模块: 进行同步时, 先检测负载主机状态, 如果可用, 则进行信息同步, 否则终止同步, 并监听负载主机可用状态, 如果连续多次检测负载机不可用的, 向备用机发起接管指令. 当新的负载主机被启用或原负载主机恢复正常, 则重新发起同步, 见算法 2.

算法 2. 交易同步算法

```
While (true)
  Foreach res in synQueue
    Foreach server in servers
      Send res to server and return synresult
    If(!synresult)
        Add res to synQueue
      End If
    End Foreach
  End foreach
End while
Returns
```

交易接管模块: 当负载主机发生异常且在设定的时间内没有正常恢复时, 新的主机被启用并接管故障主机的交易. 当主机向负载机同步失败后, 向负载机发起检测信息, 如果收到的信息异常, 重复发生检测信息, 当进行过 N 次检测后异常的, 由负载主机向备用机发起交易接管指令. 备用机接受到接管指令后, 进行交易引擎初始化, 并向当前的交易主机发起会话同步, 内核同步请求, 最后对请求结果进行校验和处理, 同步成功后, 将接管结果信息通知交易主机, 双方确认后, 新的负载机和交易主机又互为负载交易主机, 接管示意见图 7. 图中发生故障的交易服务器是 B 服务器, 为方便描述, 将 A 服务器称为交易主机, 简称 MS, B 服务器成为负载机, 简称 BS, C 备用机称为接管服务器, 简称 SPS. 接管过程主要包括两部分: 1) 算法 3 所示的异常

检测及备用机启动; 2) 算法 4 所示的备用机进行交易接管.

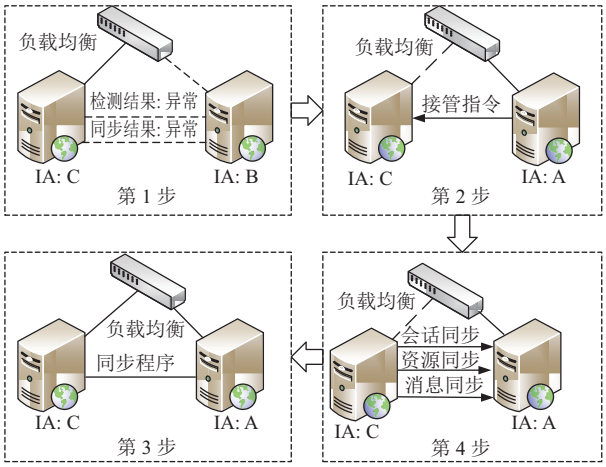

图 7 交易接管示意

交易引擎自动净化模块: 该模块的功能是检测交易系统内核状态, 释放交易资源, 重置异常线程. 设计目的是为交易系统提供一种自动净化机制, 保证交易系统资源的高可用性, 部分过程见算法 5.

算法 3. 交易接管算法-异常检测及备用机启动

```
Input: Syn Result, N
If Syn Result == false
  LN ← 1
  While (N >= LN++)
    BSMS ← GetSign From BS
    If(BSSMS)
    Return
    End If
  End While
  Update BASIT
  Send BS take over message to SPS
End If
Returns
```

算法 4. 交易接管算法-备用机进行交易接管

```
Input: BS take over message
If (BS take over)
  Init trans engine
  Syn session with MS
  Syn res with MS
  Syn check
  Update BASIT
End If
Returns
```

算法 5. 交易引擎自动净化算法

```
While (true)
  Foreach thread in thread pool
    If(thread status is exception)
      add new_thread to pool
      add task to new_thread
      release thread
    End if
  End Foreach
  Foreach resource pool
    If resource has terminate
      Remove resource from pool
      Else if resource status exception
      Get new_resource from database
      Add new_resource to pool
  End Foreach
  End While
...
Returns
```

3.3 B/S 数据交互优化

B/S 数据交互采用成熟的 AJAX 请求方式, 以降低每次请求及应答的数据量, 并根据具体的业务特点进行了进一步的优化. 以交易时间为例, 在交易过程中, 为了保证竞买人实时看到服务器时间且竞买人看到的服务器时间不至于“跳秒”, 客户端一般会在每隔不到 1 s (本例 500 ms) 的时间就向服务器端发送 1 次获取服务器时间的请求. 当参与的竞买人较多时, 系统需要接收大量的类似请求并做出应答, 给服务器带来很大的负载压力. 为解决这一问题, 本文提出“一次获取, 多次使用”的方式进行优化.

具体操作是, 以向服务器请求获取时间结合浏览器计算的方法, 来降低向服务器请求时间的频次, 例如每隔 5 分钟向服务器请求一次时间服务, 请求成功后 5 分钟内便不再向服务器发起服务器时间获取请求, 而是在浏览器端用计时器替代. 5 分钟后再次向服务器端获取服务器时间, 如此反复进行, 见算法 6.

算法 6. 服务器时间算法

```
Input: LMT
Output: New Time
LTI ← LMT
While(true)
  If ++LTI >= LMT
    LTI ← 0
    New Time ← Get Server Time
```

```
  Else
    New Time += 1
  End If
End While
returns New Time
```

4 实验及效果

按照本文的设计方法对系统进行优化后分别在江西、重庆等地将系统迁移至云平台, 并进行了测试和生产运行. 测试及运行过程中重点对 SA-IA 数据交换可靠性, 交易接管, 交易引擎自动净化, B/S 数据交换优化等几个方面进行了测试和对比统计分析. 实验数据和实际运行效果表明, 优化的系统在性能, 安全性等方面都有不同程度的提高, 采用新架构的系统能够方便的向云平台进行移植.

4.1 实验设计

系统采用 Java 语言开发, 中间件采用 Weblogic, 数据库采用 Oracle. 单元测试工具为 JUnit, 压力测试工具使用 Load Runner, 见表 1.

4.2 结果分析

交易接管方面: 对交易自动接管进行 5 个批次, 每个批次共进行 100 次接管实验的测试, 测试的主要指标是接管的成功率, 接管耗时. 通过测试, 系统的自动接管成功率是 100%, 接管耗时平均约为 22 s, 见表 2.

表 1 系统测试及运行工具

工具名	简介
Java	Java编程语言是一种通用、并行, 基于类, 面向对象的语言. Java应用程序可运行在任何运行Java虚拟机的机器上, 和具体的操作系统平台无关.
Weblogic	WebLogic是Oracle公司推出的一款应用服务中间件, 是用于开发、集成、部署和管理大型分布式Web应用、网络应用和数据库应用的Java应用服务器.
Oracle Database	Oracle Database, 是Oracle公司的一款关系数据库管理系统. 其可移植性好、功能强大, 是一种高效率、可靠性好的、适应高吞吐量的数据库解决方案.
Load Runner	Load Runner可适用于各种体系架构的自动负载测试, 能预测系统行为并评估系统性能.
Junit	JUnit是由Erich Gamma和Kent Beck编写的一个回归测试框架. Junit测试是程序员测试, 即所谓白盒测试.

表 2 自动接管测试情况统计表

	P1	P2	P3	P4	P5
接管成功率(%)	100	100	100	100	100
接管耗时(s)	25	23	19	21	22

交易引擎自动净化机制方面: 随着交易时间的推移, 引入交易引擎自动净化机制前的交易系统的连接数、JVM 内存、线程等资源的可用情况会出现连续下降. 引入后系统把即将进行的交易资源预装入至交易引擎, 交易结束后将其移除, 并实时对交易引擎进行监控, 自动重置出现异常的线程、交易资源等. 引入交易自动净化机制后, 交易系统变得更加的稳定了, 也能获得更高的性能. 采用自动净化之前和之后生产环境各半年的随机采样数据进行对比分析, 见图 8 和图 9. 可以看出, 引入自动净化机制后, 系统可用资源并不会随着交易的日积月累发生大幅减少.

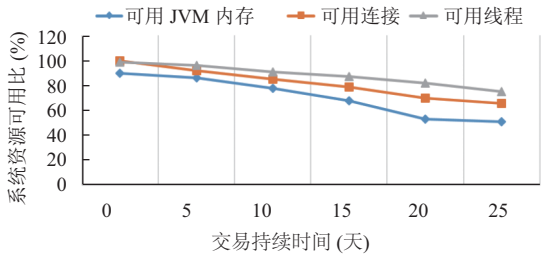

图 8 无净化机制系统的资源可用比变化情况

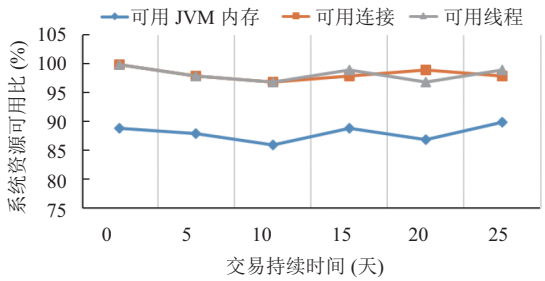

图 9 自动净化后系统的资源可用比变化情况

B/S 数据交互优化方面: 通过对服务器时间获取、资源信息获取等 B/S 数据交互优化后, 相同交易情况下, 单台服务器负载压力有了明显减轻. 为验证优化结果, 采用 HP load runner 对优化前后的访问情况进行 4 个批次的对比测试分析, V-User 数量分别为 50、500、1000、2000, 思考时间为 30 s, 每个 V-User 报价总数为 100 人次.

通过对服务器访问量进行统计, 见表 3. 可以看出, 在相同交易量情况下, 优化后服务的访问量较优化前下降了约 67.5%, 主要是因为访问方式优化后, 降低了向服务器的请求频次, 使得在服务器性能相当的情况下, 单台服务器能承担更多的交易任务.

表 3 B/S 交互优化前后服务器访问量对比

实验批次	P1	P2	P3	P4
出让资源	10	100	100	200
竞买人总数	50	500	1000	2000
持续时间(s)	1586	15 090	31 831	63 729
优化前	377 965	25 087 518	101 871 968	228 358 762
优化后	123 015	8121 756	32 965 180	73 929 260
访问量下降(%)	67.45	67.63	67.64	67.63

5 结束语

得益于云平台的成本节约、资源共享便利性等方面的优势, 目前越来越多的传统应用正在或即将向云平台迁移. 而传统的应用系统由于其业务各自差异较大, 部署方式、数据交换方式、访问方式等都各有其自身特点. 如何高效地进行应用迁移; 如何保证系统迁移后的例如性能、稳定性、安全性不贬值, 都需要在应用迁移前进行认真的思考研究. 本文采用以规划、执行、评估为核心迁移过程框架, 结合系统自身特点, 制定出了合适了解决框架, 并设计了主要的算法, 并进行了实现. 最后分别在江西、重庆等地进行了实地测试和生产运行, 实践表明, 迁移后系统运行稳定, 性能表现出色. 这种迁移方法, 可以为已有行业应用系统向云平台迁移提供参考, 例如国土、公安、城市规划等部门.

参考文献

1 张靓, 范冰冰, 郑伟平. 云平台应用系统迁移方法的研究. 计算机科学, 2013, 40(6A): 271–275.

2 Andrade PRM, Araújo RG, Filho JC, *et al*. Improving business by migrating applications to the cloud using cloudstep. Proceedings of the 29th International Conference on Advanced Information Networking and Applications Workshops. Gwangiu, South Korea. 2015. 77–82.

3 García-Galán J, Trinidad P, Rana OF, *et al*. Automated configuration support for infrastructure migration to the cloud. Future Generation Computer Systems, 2016, (55): 200–212. [doi: 10.1016/j.future.2015.03.006]

4 Yousif M. Migrating applications to the cloud. IEEE Cloud Computing, 2016, 3(2): 4–5. [doi: 10.1109/MCC.2016.42]

5 Dhuria S, Gupta A, Singla RK. Migrating applications to the cloud: Issues and challenges. International Journal of Advanced Research in Computer Science and Software Engineering, 2015, 5(6): 958–961.

6 Scandurra P, Psaila G, Capilla R, *et al*. Challenges and assessment in migrating IT legacy applications to the cloud. Proceedings of the 9th Maintenance and Evolution of Service-Oriented and Cloud-Based Environments. Bremen, Germany. 2015. 7–14.

7 Tankovic N, Grbac TG, Truong HL, *et al*. Transforming vertical Web applications into elastic cloud applications. Proceedings of 2015 IEEE International Conference on Cloud Engineering. Tempe, AZ, USA. 2015. 135–144.

8 Jamshidi P, Ahmad A, Pahl C. Cloud migration research: A systematic review. IEEE Transactions on Cloud Computing, 2014, 1(2): 142–157.

9 Gholami MF, Daneshgar F, Low G, *et al*. Cloud migration process-A survey, evaluation framework, and open challenges. Journal of Systems and Software, 2016, (120): 31–69. [doi: 10.1016/j.jss.2016.06.068]

10 Hwang J, Bai K, Tacci M, *et al*. Automation and orchestration framework for large-scale enterprise cloud migration. IBM Journal of Research and Development, 2016, 60(2-3): 1:1–1:12. [doi: 10.1147/JRD.2015.2511810]

11 赵俊三, 丁强龙, 陈国泉. 国土资源网上交易系统内核优化设计研究. 昆明理工大学学报 (自然科学版), 2015, 40(1): 29–34.