# 基于 SWIFT 的海量数据存储平台设计

李朋远[1] 张志勇[2]
(北京跟踪与通信技术研究所 北京 100089)[1] (太原卫星发射中心 太原 030000)[2]

**摘 要** 随着我国航天事业的迅速发展,利用海量数据存储平台存储产生的海量科研数据变得越来越重要。文中提出了一种基于 SWIFT 分布式存储系统的云存储方案,并构建了存储平台的基础架构。存储平台中 SWIFT 系统的设计主要包含数据存储的哈希过程、Ring 优化、Partition 优化和 Replica 副本决策 4 部分,通过数据模拟的方式验证了关键设计的有效性。

**关键词** SWIFT,Ring,Partition,Replica,分布式存储
**中图法分类号** TP393.07 **文献标识码** A

## Design of Storage Platform for Large Scale Data Based on SWIFT System

LI Peng-yuan[1] ZHANG Zhi-yong[2]
(Beijing Institute of Tracking and Telecommunications Technology, Beijing 100089, China)[1]
(Taiyuan Satellite Launch Center, Taiyuan 030000, China)[2]

**Abstract** With the rapid development of China's space activities, storing the massive data based on the huge data storage platform becomes increasingly important. This paper presented a cloud storage solution based on one distributed storage system which is named SWIFT, and built the infrastructure architecture for this storage platform. The design of SWIFT mainly includes four parts, the hash process of data storage, Ring, Partition, and Replica policy. And this paper verified the validity of the key design of SWIFT through the way of data simulation.

**Keywords** SWIFT, Ring, Partition, Replica, Distributed storage

## 1 引言

"天宫二号"与"神州十一号"的成功发射标志着我国探索外空的脚步又向前迈近了一步。随后我国将进一步加快对外太空的探索,届时长期的在轨运行将产生大量的航天科研数据。同时相关的科研单位在日常的研究中也会产生大量的实验数据。这些数据具有海量、复杂、多样、异构等特性,研究如何将这些数据进行统一存储、科学管理具有重要意义,对未来的科学实验具有重要的参考价值。

通常使用 SAN 存储网络和高性能存储设备相结合的方式来存储结构化大规模数据,并且通过热双工的方式保证数据可靠性,这需要大量的经费购买存储资源[1]。互联网思维不断地影响并推动着科技的发展,海量数据分布式存储技术已成为解决互联网使用过程中产生的海量非结构化数据的存储问题的重要方法[2]。自 2012 年以来,OpenStack[3] 已成为业界最受关注的开源云计算平台[4]。沃尔玛于 2014 年 8 月将其全部电子商务业务迁移到 OpenStack,并于同年底通过该平台实现支撑所有美国流量。国外汽车巨头宝马、视频巨头华纳,国内包括百度、携程、京东、阿里巴巴、高德等互联网巨头企业均通过 OpenStack 平台部署相关业务[4]。世界瞩目的天河二号超级计算机通过该平台部署其 HPC 云环境[4]。

SWIFT[5] 作为 OpenStack 云计算平台的子项目之一,为整个平台提供存储服务,其强大的扩展性、冗余性和持久性保证了平台各项业务的高效运行[5-6]。

本文以 SWIFT 分布式云存储技术为基础,提出了一种海量科研数据的云存储方案,构建了基于 SWIFT 的云存储平台,并深入研究了 SWIFT 分布式存储的总体架构和关键设计。

## 2 云存储总体方案

SWIFT 存储系统中元数据和对象数据的存储均采用完全随机分布机制,保证了元数据的存储不会成为整个系统的瓶颈[5-6]。并且整个 SWIFT 集群可以部署为完全无单点,SWIFT 本身的设计架构保证了系统不存在单点业务[5]。

本文根据航天系统的数据特点,通过 SWIFT 存储系统实现存储资源的整合、数据到存储节点的存储、数据的备份以及存储容量的调配,提出了云存储方案(见图 1),以达到实现航天系统所有数据集中存储管理的目的。

所提方案将所有的软件应用后台部署在应用服务集群;Web 网络应用后台部署在 Web 服务集群;数据库服务器集群提供所有的数据库请求服务,通过去异构化处理技术,将现有的关系型数据库系统整合到该云存储平台中[7];数据归档保存功能由备份服务集群提供支持。

所提方案以安全、稳定、不间断地为航天系统提供海量数据存储服务为根本目的,使得各种科研数据得到妥善的存储管理,摆脱原有的刻盘归档保存或者数据拷贝复制保存的模式,不同单位间也可以通过综合信息显示系统、信息管理系统等平台进行信息共享互联,以更好地发挥航天科研数据的价

李朋远(1989-),男,硕士,主要研究方向为计算机系统总体设计、高性能计算,E-mail:pengyuan_li@163.com;张志勇(1988-),男,主要研究方向为计算机系统总体设计。

值，为航天事业的进一步发展服务。

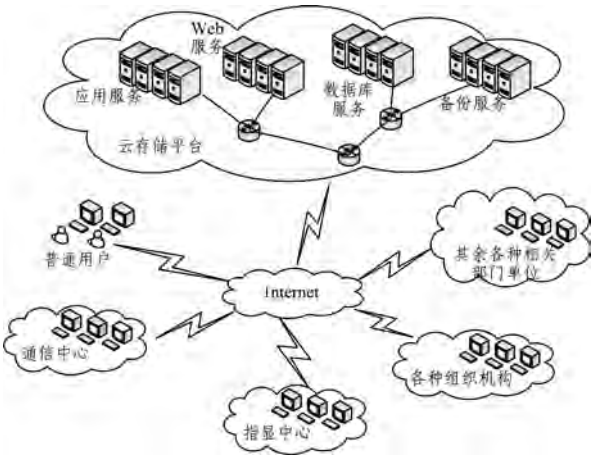



图 1 云存储总体方案

## 3 基于 SWIFT 系统的存储平台开发

基于 SWIFT 分布式存储系统的存储平台可以接收用户的数据读写请求并及时响应。SWIFT 系统处理数据存储业务，实现对海量科研数据的存储管理[5-6]。

### 3.1 存储平台基础架构

本文构建的云存储基础架构采用经典的 MVC[8] 3 层架构模式，如图 2 所示。该架构具有设计结构清晰、系统易于扩展的特点。

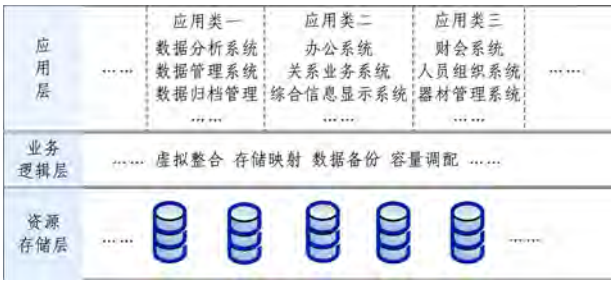



图 2 平台基础架构

(1)应用层(视图层)：为用户提供视图，方便用户与服务后台交互，具有易用、友好的特点。其中，应用类一面向管理人员开放，可以进行分析、管理、归档数据等类似操作；应用类二面向一般用户开放，主要包括日常办公系统、业务关系维护系统、综合信息显示系统等；应用类三面向特定的组织机构开放，包括财会系统、器材管理系统、人员组织关系系统等。

(2)业务逻辑层：其包含两层含义。首先，完成对数据业务的处理、请求类别判定、响应数据库处理等业务；其次，利用存储虚拟化技术进行存储资源整合，实现数据的高可用存储以及存储节点的有效管理。

(3)资源存储层：其为整个架构的基础，是实际的数据存储池，实现对海量数据的存储。

### 3.2 SWIFT 系统总体架构

SWIFT 总体架构主要包括以下组成部分(见图 3)：

(1)Load Balance 负载均衡器：以配置策略为依据，将客户端的请求分配到无状态的 Proxy Node。

(2)Proxy Node 代理节点：通过 SWIFT API 的服务进程转发客户端的请求。

(3)Storage Node 存储节点：通过 SWIFT 中的存储服务将数据存储到物理磁盘上。按照存储目标的不同类型可将 Storage Node 上运行的存储服务分为以下 3 类。

1)Object Server：对象存储服务，为对象提供存储服务。

2)Container Server：容器存储服务，为对象列表提供存储服务。

3)Account Server：账户存储服务，为容器列表提供存储服务。

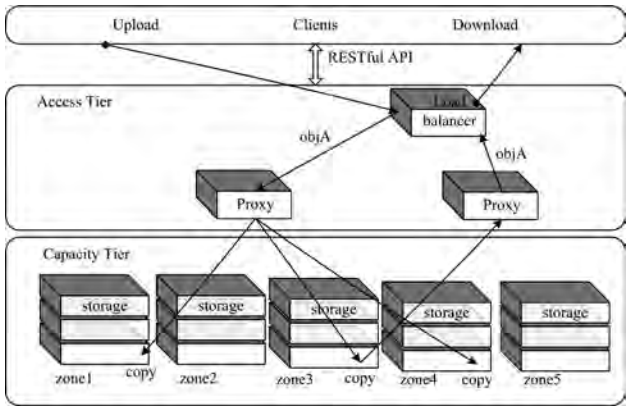



图 3 SWIFT 总体架构示意图

### 3.3 SWIFT 系统的关键设计

存储系统中，哈希算法是实现数据到物理存储节点的基础与关键[9]。以哈希算法为基础，SWIFT 通过 Partition(虚节点)和 Ring(环)实现了数据的高效存储，Partition 的引入可以将整个集群的存储空间拆分为若干个存储点(与存储节点不是同一概念)，并通过 Ring 实现 Partition 到磁盘上的物理存储点的映射。replica 复制进程则保证 Partition 副本的有效分布及数据的安全。

单个节点的负载过重会引起整个存储系统的崩溃[11]。本节通过算法分析的方式详细分析 Ring 如何在 Partition 映射到物理存储节点的过程中保证存储的负载均衡，以及如何保证存储容量的线性增长。

#### 3.3.1 基于哈希算法实现数据存储

一般情况下，通过哈希算法完成对象到存储节点的映射[8]。该过程主要包括两个步骤：

(1)计算对象的哈希值 Key；

(2)计算 Key mod $N$ 值 $C$，$N$ 表示一个特定集群中存储节点的个数。

按照此过程，数据对象会被平均哈希到 $N$ 个存储节点中，$C$ 值为对应对象应该存放的节点号。集群节点在数据访问量均衡时负载均衡。然而，这样的哈希分配在存储扩容时存在严重缺陷。

按照上述哈希过程，节点数 $N$ 为 2 时，Key 为 0,1,2,3,4 的对象分别存放在 0,1,0,1 和 0 号节点上。当两个节点无法满足存储需求而新增一个存储节点时，$N$ 变为了 3。根据上述算法，Key 值为 2,3,4 的数据对象需要按照 2(节点 0->节点 2)、3(节点 1->节点 0)、4(节点 0->节点 1)进行数据迁移。如果集群中已经存储大量甚至海量数据，那么数据迁移将占用很长时间。

扩大集群及集群中存储数据的规模，按照上述哈希过程将 107 项数分配到 100 个集群节点，假设某数据对象的哈希值为 $X$，那么：

满足条件 $X$ mod (100)==$X$ mod (101)的数据无需迁移；

满足条件 $X$ mod (100) !=$X$ mod (101)的数据必须迁移。

通过计算发现，增加一个存储节点提供存储服务时需要移动 9900989 个数据项，即 99.01%的数据项。由此可知，如果按照上述算法实现对象到存储节点的映射，当集群节点数 $N$ 变大，增加存储节点提升集群存储能力时，整个存储系统中几乎所有的数据都将被迁移，从而严重影响系统的性能和扩展性。

3.3.2　通过 Ring 减少数据迁移

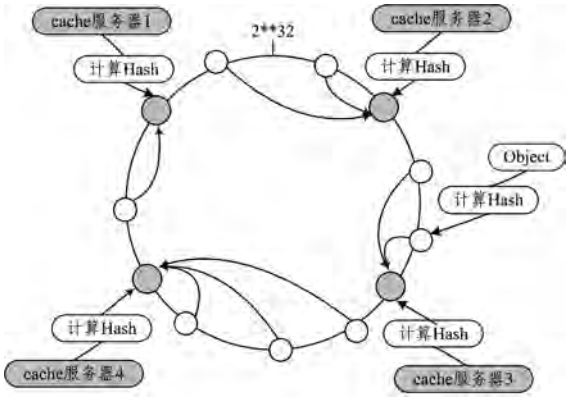
为了提升系统的性能，SWIFT 首先通过引入 Ring 结构来解决存储节点增加过程中数据迁移量过大的问题。具体步骤如下：

(1)将所有的节点(按照机器名或 IP 地址等)哈希到一个封闭的圆环上，圆环可以理解为术语“Ring”；

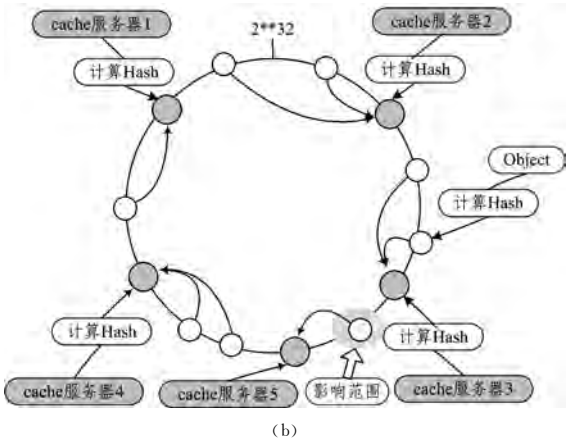
(2)将所有的数据对象哈希到步骤(1)中的圆环上；

(3)按照顺时针方向，以对象映射到的位置为基准点进行查找，将对象保存到查找到的第一个节点上。

图 4(a)给出了上述 Ring 哈希空间的分布示意图。与 3.3.1节中的哈希过程相比，数据对象到物理存储节点的存储方式发生了根本性的变化。




(a)




(b)

图 4　基于 Ring 环的数据移动示意图

在图 4(a)对应的存储集群中增加一个存储节点，假设为 cache5，按照上述过程将 cache5 映射到圆环上，假设位于 cache3 和 cache4 之间，那么原存储于 cache4 而在圆环上位于 cache3 和 cache5 之间的数据需要被迁移到 cache5 上，如图 4(b)所示。该过程并没有考虑集群负载均衡的问题。为了简化计算过程，假设有 100 项数据，在 cache5 增加前后，各节点上存储的数据项的范围、需要移动的数据项以及移动个数如表 1 所列。通过计算可知，若增加 25%的存储容量，则需要移动 35%的对象。

表 1　新增节点的影响范围

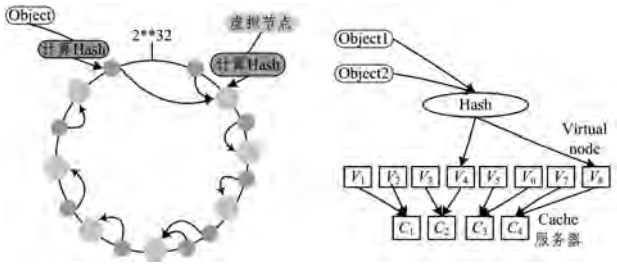
| 节点号 | 1 | 2 | 3 | 5(新增) | 4 |
|---|---|---|---|---|---|
| 原数据项范围 | 0～24 | 25～49 | 50～74 | | 75～99 |
| 新数据项范围 | 0～19 | 20～39 | 40～59 | 60～79 | 80～99 |
| 需移动数据项 | 20～24 | 40～49 | 60～74 | | 75～79 |
| 需移动个数 | 5 | 10 | 15 | | 5 |

按照 3.3.1 节的集群和数据规模，首先将 107 个数据对象按照上述算法哈希到 100 个节点的集群中，然后增加一个节点提升集群存储能力，按表 1 的计算方法进行计算(过程省略)，最后大约需要移动 50%的数据。

由分析可知，按照本节描述的算法进行数据到物理存储节点的映射，提高集群 1%的存储能力需要移动大约 50%的数据。与 3.3.1 节的计算结果相比，系统性能得到了提升，但仍不理想。

3.3.3　通过 Partition 减少数据迁移

为了进一步提升系统的性能，SWIFT 系统在 Ring 环的基础上引入了 Partition 结构，数据对象、Partition、物理节点间的映射关系如图 5(b)所示。3.2 节中描述的对象存储服务、容器存储服务、账户存储服务中对应的数据对象、容器数据库、账户数据库均以 Partition 为最小单位在物理存储节点进行存储。Partition 仍然通过哈希分配到圆环上，如图 5(a)所示。与 3.3.2 节的哈希过程相比，此处通过 Ring 实现数据对象到物理存储节点存储的过程中，Partition 替代了原数据对象的位置。




(a)虚拟节点排列分布示意　　(b)object，vnode，node 映射关系

图 5　虚拟节点引入示意图

按照最初的假设，将 107 项数据对象映射到集群中的 100 个存储节点上。首先引入 1000 个 Partition 节点，在负载均衡的条件下每个节点会被分配 10 个 Partition 节点。为了提升系统的存储能力，增加一个存储节点，那么在集群负载均衡的约束下就需要从存储节点中的 9 个节点上分别迁移一个 Partition 到新增加的存储节点上，也就是说共移动 9 个 Partition 节点，即 90000 项数据。

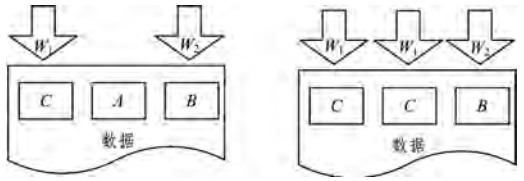
由分析可知，按照本节算法实现数据到物理存储节点的映射，提高集群 1%的存储能力需要移动 0.9%的数据。与 3.3.1 节和 3.3.2 节中的算法相比，系统性能得到了较大的提高。

3.3.4　基于 NWR 策略的 Replica 副本决策

3.3.1 节－3.3.4 节解决了数据在集群中的单节点存储问题、节点间的负载均衡问题以及集群扩容过程中的数据迁移问题，本小节重点解决数据安全问题。SWIFT 系统以 NWR 策略为依据，通过引入 Replica 副本来保证数据的安全。

为体现 NWR 策略的重要意义，首先做如下假设：SWIFT 系统中的某个数据对象 $P$ 的副本数为 3，且每个副本的值都为 $A$。

SWIFT 分布式存储系统通常会处理很多的并发读写请求。如果存在进程 $W_1$ 和 $W_2$ 同时请求数据对象 $P$ 的写操作，且 $W_1$ 请求写入的值为 $C$，$W_2$ 请求写入的值为 $B$，在没有任何条件约束的情况下，$W_1$ 和 $W_2$ 写入数据对象 $P$ 的结果可能如图 6(a)所示，各自进程面向不同的对象副本执行了一次写操作，且执行成功后各自返回，最终导致同一个数据对象的 3 个副本值都不同。




(a) (b)

图 6 并发写数据

此时，如果存在一个进程请求对象 $P$ 的读操作，该进程的读取结果可能如图 7(a)所示，读取到的数据为过时的甚至是很久以前未更新的数据。

SWIFT 系统为了保证并发读写数据的正确性，引入了 NWR 策略，具体含义如下[10]。

(1)$N$：一个对象在集群中存储的副本份数；

(2)$W$：对一个对象执行写操作时，需要确保成功写副本的份数；

(3)$R$：对一个对象执行读操作时，需要读取副本的份数。

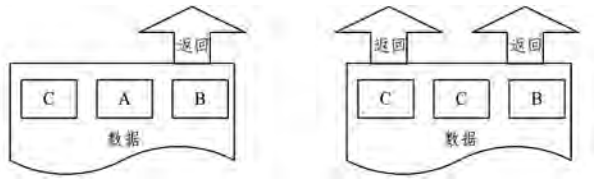
并且上述 3 个值需要满足以下两个条件[10]：

(1)$W>N/2$，两个事务不能并发地写同一个对象；

(2)$W+R>N$，对某一个对象进行读或写操作时，所读的副本集合与写的副本集合必须有交集。

条件(1)为执行写请求进行约束。进程执行写请求成功的判定依据是写副本的个数大于所有副本数的一半，否则系统认为执行写操作失败。可以将条件(1)看作写的锁，解决了写副本竞争问题。按照条件(1)，上面例子中写的副本数至少为 2，$W_1$ 和 $W_2$ 个两进程执行写操作的结果如图 6(b)所示，$W_1$ 执行成功时，$W_2$ 执行失败。

条件(2)为执行读请求进行约束。该条件等价于 $R>N-W$，对于上面的例子，进程要读取的副本数至少为 2，此时进程读取数据的结果如图 7(b)所示。条件(2)有效地避免了进程只读取到过时的数据，保证进程获取的数据中至少有一个是最新的，进而可以通过别的判定方法从多个读取到的数据中获取正确的数据。




(a) (b)

图 7 并发数据读取

## 4 关键技术验证

本节通过数据模拟的方式，验证第 3 节分析的 SWIFT 关键设计算法。所有代码通过 Python 实现，如图 8 所示。

代码 A：普通哈希代码

```
NODE_COUNT=100
NEW_NODE_COUNT=101
DATA_ID_COUNT=10000000
MOVED_IDS=0
for data_id in xrange(DATA_ID_COUNT):
  data_id=str(data_id)
  hsh=unpack_from('>/',md5(data_id).digest())[0]
  node_id=hsh % NODE_COUNT
  new_node_id=hsh % NEW_NODE_COUNT
  if node_id != new_node_id:
    MOVED_IDS += 1
percent_moved=100.0 * MOVED_IDS/DATA_ID_COUNT
```

代码 B：SWIFT 引入环并考虑负载均衡后哈希代码

```
NODE_COUNT=100
NEW_NODE_COUNT=101
DATA_ID_COUNT=10000000
MOVED_IDS=0
node_range_starts=[]
new_node_range_starts=[]
for node_id in xrange(NODE_COUNT):
node_range_starts.append(DATA_ID_COUNT/NODE_COUNT *
                         node_id)
for new_node_id in xrange(NEW_NODE_COUNT):
  new_node_range_starts.append(DATA_ID_COUNT/NEW_
                               NODE_COUNT * new_node_id)
for data_id in xrange(DATA_ID_COUNT):
  data_id=str(data_id)
  hsh=unpack_from('>/',md5(data_id).digest())[0]
  node_id=bisect_left(node_range_starts,hsh%DATA_ID_
          COUNT) % NODE_COUNT
  new_node_id=bisect_left(new_node_range_starts,hsh%DATA_
              ID_COUNT) % NEW_NODE_COUNT
  if node_id != new_node_id:
    MOVED_IDS += 1
percent_moved=100.0 * MOVED_IDS/DATA_ID_COUNT
```

代码 C：SWIFT 引入虚节点后哈希代码

```
NODE_COUNT=100
DATA_ID_COUNT=10000000
VNODE_COUNT=1000
MOVED_IDS=0
vnode2node=[]
for vnode_id in xrange(VNODE_COUNT):
  vnode2node.append(vnode_id%NODE_COUNT)
new_vnode2node=list(vnode2node)
new_node_id=NODE_COUNT
vnodes_to_assign=VNODE_COUNT/(NODE_COUNT+1)
while vnodes_to_assign > 0:
  for node_to_take_from in xrange(NODE_COUNT):
    for vnode_id,node_id in enumerate(vnode2node):
      if node_id == node_to_take_from:
        vnode2node[vnode_id]=new_node_id
        vnodes_to_assign -= 1
        if vnodes_to_assign <= 0:
          break;
    if vnodes_to_assign <= 0:
      break;
for data_id in xrange(DATA_ID_COUNT):
  data_id=str(data_id)
  hsh=unpack_from('>/',md5(data_id).digest())[0]
  vnode_id= hsh%VNODE_COUNT#(1)
  node_id=vnode2node[vnode_id]
  new_node_id=new_vnode2node[vnode_id]
  if node_id != new_node_id:
    MOVED_IDS += 1
percent_moved=100.0 * MOVED_IDS/DATA_ID_COUNT
```

图 8 SWIFT 关键技术代码验证

根据第 3 节中的例子，NODE_COUNT 表示 100 个集群节点，DATA_ID_COUNT 表示 107 项数据，代码运行结果 percent_moved 表示增加一个节点时的数据移动量，普通哈希下增加一个节点时的数据移动量为 99.01%，引入 Ring 环后增加一个节点时的数据移动量为 49.02%，引入 Partition 虚节点后增加一个节点时的数据移动量为 0.90%。测试结果

与第 3 节中分析的计算值一致。

**结束语**　本文提出了一种基于 SWIFT 分布式存储系统的云存储方案，构建了云存储平台的基础架构，对 SWIFT 存储系统的总体架构进行了分析，进而详细地对 SWIFT 系统中的关键设计进行了算法分析。SWIFT 本身的架构保证了集群的存储无单节点和无单点业务的特性。SWIFT 系统中 Ring 和 Partition 的引入使得搭建 SWIFT 系统的存储平台在存储容量扩充时数据迁移量最小。基于 NWR 策略的副本决策方法保证了数据的安全性。本文通过实验验证了 SWIFT 关键设计算法的有效性，结果也表明 SWIFT 系统具有良好的扩展性。总体来说，基于 SWIFT 系统设计海量数据存储平台对未来航天系统的发展具有一定的意义，但还需要实际的应用环境来检验。

## 参 考 文 献


[1] 刘鹏. 云计算——将计算变成水和电[J]. 中国计算机学会通讯，2009，5(10)：49-54.

[2] WEN X，GU G，LI Q，et al. Comparison of open－source cloud management platforms：OpenStack and OpenNebula[C]// 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD). IEEE，2012：2457-2461.

[3] GUPTA R. Above the Clouds：A view of cloud computing[J]. Eecs Department University of California Berkeley，2009，53(4)：50-58.

[4] 陆平，赵培，左奇. OpenStack 系统架构设计实战[M]. 北京：机械工业出版社，2016.

[5] ARNOLD J. OpenStack Swift：Using，Administering，and Developing for Swift Object Storage[M]. O'Reilly Media，Inc.，2014.

[6] TOOR S，TOEBBICKE R，RESINES M Z，et al. Investigating an open source cloud storage infrastructure for CERN-specific data analysis[C]// 2012 IEEE 7th International Conference on Networking，Architecture and Storage (NAS). IEEE，2012：84-88.

[7] 李俊，李勇. 联邦式异构数据库应用系统的集成框架和实现技术研究[J]. 计算机应用研究，2001，18(4)：19-22.

[8] 钟茂生. 软件设计模式及其使用[J]. 计算机应用，2002，22(8)：32-35.

[9] LEWIN D M. Consistent hashing and random trees：algorithms for caching in distributed networks[D]. Massachusetts Institute of Technology，1998.

[10] 武志学，赵阳，马超英. 云存储系统——Swift 的原理、架构及实践[M]. 北京：人民邮电出版社，2015.

[11] GODFREY B，LAKSHMINARAYANAN K，SURANA S，et al. Load balancing in dynamic structured P2P systems[C]// Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004). IEEE，2004：2253-2262.


(上接第 597 页)

复实验，实验中机械臂的运动速度、操作范围不变，传送带的最大速度为 200 mm/s，系统对不同的餐具目标进行检测识别，并通过 Scara 机器人实现分拣。系统实验结果如表 1 所列。

表 1　实验结果

| 传送带速度/(mm/s) | 抓取总数 | 有效识别 | 漏抓个数 | 误抓个数 |
|---|---|---|---|---|
| 100 | 100 | 100 | 0 | 0 |
| 120 | 100 | 100 | 0 | 0 |
| 150 | 100 | 100 | 2 | 0 |
| 180 | 100 | 100 | 1 | 0 |
| 200 | 100 | 100 | 2 | 0 |

由结果可以看出，系统的平均识别抓取率为 99%，满足机器人高速、高精度的特点，可以快速、准确地完成餐具目标的自主分拣操作。实验中出现漏抓的原因是放置餐具时，物件密度过大，导致机器人在自动匹配计算中选择放弃抓取。

**结束语**　在给定的条件下，本文系统地分析了机器视觉的结构和各个组成部分的作用和功能。通过分析碗碟的表面反光特性及系统实际工作条件，选择了恰当的光源，并结合软件部分的编写及各个打光方式的比较，给出了系统合适的打光方式。通过比较，分析并给出了一种适合于本系统的基于像素当量的摄像机标定方法，最后在 LabVIEW 环境下利用机器视觉技术设计了一个运动餐具分拣系统，呈现了机器视觉技术在餐具分类中的应用。整个系统在 TCP/IP 协议下通信，上位机相机采集图像，经过相机标定、图像预处理、目标识别跟踪等算法计算餐具目标的运动状态并分析类别，根据餐具的运动状态控制机械手对餐具进行“抓取”。实验结果表明，所提系统可以连续、快速地抓取流水线上的运动餐具，并将不同类别的餐具按指定要求分类“放置”于不同区域。后期还需要对系统做进一步的改进，系统跟踪识别目标定位的信息仅来源于工业相机，可以在传送带上加入旋转编码器，时刻记录目标当前的位置信息，与视觉位置信息综合考虑，提高系统的精度和鲁棒性。此外，可以进一步优化控制算法，从而减少动态误差，加强对目标跟踪的能力，提升系统的整体性能以及实用性。

## 参 考 文 献


[1] 贾云得. 机器人视觉[M]. 北京：科学出版社，2000.

[2] NI Vision Concepts Manual[M]. United State：NI，2005.

[3] 唐向洋，张勇，李江有，等. 机器视觉关键技术的现状及应用展望[J]. 昆明理工大学学报(理工版)，2004，29(2)：5-7.

[4] ADELSON E H，BERGEN J R. The plenoptic function and the elements of early vision. Computational Models of Vision Processing [M]. Cambridge MA：MIT Press，1991.

[5] 殷焰，赵荣椿. 一种新的基于直线特征的摄像机自标定方法[J]. 计算机应用研究，2006，23(3)：170-171.

[6] 刘焕军，王耀南，段峰. 机器视觉中的图像采集技术[J]. 电脑与信息技术，2003(1)：20-23.

[7] 裴忠发. 基于 LabVIEW 的机器视觉实现[J]. Meehanieal & Electircal Engineierng Magazsne，2002，19(4)：53-55.

[8] 马颂德，张正友. 计算机视觉-计算理论与算法基础[M]. 北京：北京科学出版社，2003：52-93.

[9] 刘鹏飞，韩九强，段延礼，等. 基于开放式控制器的机器人视觉伺服系统研究[J]. 计算机集成制造系统，2006，12(6)：955-960.

[10] 曹健. 图像目标的表示与识别[M]. 北京：机械工业出版社，2012.

[11] 李国栋. 基于图像雅可比矩阵的关节机器人视觉伺服控制系统研究[D]. 湘潭：湘潭大学，2009.

[12] 李锋. 机器视觉应用技术研究[D]. 杭州：浙江大学，2003.

[13] MARR D. A Computational Investigation into the Human Representation and Processing of Visual Information[M]// Vision：A Computational Investigation into the Human Representation and Processing of Visual Information. Henry Holt and Co. Inc. 1982：107-111.

[14] 张志远，伏冬孝，段坚. 工业机器人视觉系统的目标定位[J]. 机械工程与自动化，2013，32(5)：130-131.