# 一种基于Paxos算法的证券交易系统内存复制方法研究

黄晓东[1,2] 张 勇[1,3] 邢春晓[1,3] 黄寅飞[4] 武剑锋[4] 白 硕[4]

(清华大学信息技术研究院 北京 100084)[1] (清华大学计算机科学与技术系 北京 100084)[2]
(清华大学信息科学与技术国家实验室 北京 100084)[3] (上海证券交易所技术中心 上海 200120)[4]

**摘 要** 近年来随着高速网络技术的发展与高频交易需求的增加,提升交易速度成为电子商务交易提供者的重要关切。当前交易系统通常采用基于共享存储的主备机复制方法来保证高可用性与数据持久性,但因其存在持久化的性能瓶颈而无法进一步降低延迟。为此,提出一种基于Paxos算法的内存数据复制方法,即通过消息传递完成主备机复制,以保证结点间数据的一致性,容忍可能发生的良性故障;并以证券交易系统场景为例对其进行分析。实验结果表明,相比基于共享存储的主备机复制,该方法在万兆以太网环境下可将交易系统订单处理延迟由毫秒级降至百微秒级,并在主机故障时正确地完成热备切换。

**关键词** 数据复制,高可用性,低延迟,交易系统,Paxos算法

**中图法分类号** TP392 **文献标识码** A

## Research on a Paxos-based Approach for Memory Data Replication in Stock Trading System

HUANG Xiao-dong[1,2] ZHANG Yong[1,3] XING Chun-xiao[1,3] HUANG Yin-fei[4] WU Jian-feng[4] BAI Shuo[4]

(Research Institution of Information Technology, Tsinghua University, Beijing 100084, China)[1]
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)[2]
(Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China)[3]
(Technology Center, Shanghai Stock Exchange, Shanghai 200120, China)[4]

**Abstract** As the development of high-speed network technology and rising demand for high-frequency trading, upgrading trading speed has become the main focus of e-commerce trading system providers in recent years. Primary-backup replication based on shared storage is a classical approach to ensure high availability and data durability of trading systems at present, but it is difficult to further reduce the system latency due to its persistence bottleneck. To solve this problem, a Paxos-based approach for memory data replication was proposed and illustrated in the stock trading context. The approach accomplishes the primary-backup replication through messaging, can ensure the strong consistency of data replicas and tolerate possible benign failures. Experimental results show that compared with the shared storage replication approach, this approach reduces the order processing latency of stock trading system from several milliseconds to several hundred microseconds in 10 G-bit Ethernet, and achieves hot failover correctly in case of the primary host failure.

**Keywords** Data replication, High availability, Low latency, Trading system, Paxos algorithm

## 1 引言

近年来随着计算机硬件的发展与应用需求的变化,OLTP数据库系统出现了集群计算、内存数据驻留、单线程处理、消除事务、消除日志并通过备用组件来提供恢复能力等设计趋势[1,2]。这些趋势已对各类电子商务交易系统的设计产生了深远影响。

在各类交易系统中,证券交易系统是具有高性能、高可用性、严格约束等要求的典型OLTP应用[3],其设计中对以上趋势的体现尤为显著,可为构建类似系统提供参考。随着高频交易需求与市场容量的日益增加,证券交易系统的延迟性能近年来逐渐成为全球各大交易所提高技术竞争力的焦点,也凸显了改进OLTP架构以适应最新技术与需求发展的必要性。

证券交易系统中最为核心的组件是撮合引擎,它根据到达时间、交易价格等因素将客户提交的证券订单进行撮合成

到稿日期:2012-02-28 返修日期:2012-05-25 本文受国家科技支撑计划基金项目(2012BAH13F02,2012BAH13F04),国家自然科学基金项目(61170061)资助。

黄晓东(1987-),男,硕士生,主要研究方向为数据库技术,E-mail:huangxiaodong2008@gmail.com;张 勇(1973-),男,博士后,副研究员,硕士生导师,CCF高级会员,主要研究方向为数据库技术、数字图书馆等;邢春晓(1967-),男,博士,研究员,博士生导师,CCF高级会员,主要研究方向为数据库技术、数字图书馆等;黄寅飞 男,博士,高级工程师,主要研究方向为证券信息处理;武剑锋 男,博士,高级工程师,主要研究方向为证券信息处理;白 硕 男,博士,研究员,主要研究方向为证券信息处理。

交并返回确认消息。为保证交易系统的可用性，须对撮合引擎进行冗余备份。对撮合主机数据进行备份并保证主备机间数据的一致性，需由数据复制技术实现。具体复制方法对订单处理的延迟、吞吐量等性能指标以及数据的持久性等都具有重要影响，因此是系统设计的核心。

在证券交易系统中实现主备机复制的经典方法是基于共享存储的主备复制技术[3-5]。该方法通过访问共享存储设备中的订单成交日志文件，来实现主、备机间数据的同步以及故障下的热备切换。该方法中，订单在其成交日志持久化后才能发送确认，且共享存储设备中需对数据进行冗余备份和事务保护来确保数据的可靠性，因而存在同步持久化的性能瓶颈，难以大幅改进延迟性能。

与此同时，万兆以太网等高速网络技术在带宽与延迟等方面的持续改善，使得基于消息传递的状态机复制[6]方法相比上述基于共享存储的主备复制方法，在降低系统延迟方面更具潜力。基于消息传递的复制技术的复杂性在于如何避免因网络分区或结点故障而造成结点间数据不一致的情况，将交易数据可靠地复制到各个结点。

Paxos 算法[7,8]是一种经典的容错一致性协议，可解决上述分布式复制中保证各结点数据一致性的问题。该算法及其典型部署方式 Multi-Paxos 协议近年来被广泛应用在分布式存储系统[9-13]中来实现分布式锁服务或多副本数据复制等。但以上系统中的 Paxos 算法实现在对性能要求极高的 OLTP 系统（如证券交易系统）中还有以下一些问题：1）主要针对磁盘存储而缺乏针对内存驻留数据的高性能容错访问的通用框架；2）在 Paxos 协议过程中需对结点的协议状态进行持久化以保证故障后恢复的结点不会破坏数据的一致性，使协议性能大大受限；3）上述系统中对于主结点选举等问题未给出具体明确的解决方法。

本文针对具有极高性能要求的 OLTP 场景提出了一种基于 Paxos 算法的内存数据复制框架，并利用该框架实现了证券交易系统撮合主机内存数据的低延迟复制方法。在该方法中，通过 Multi-Paxos 协议来保证良性故障环境下结点间数据的一致性，并给出了主结点选举等问题的解决方法。为提高协议性能，该方法中消除了 Paxos 部署中对协议状态的持久化操作，并相应地给出了结点故障恢复机制，通过对更新序列的异步持久化及创建内存数据快照等机制来保证必要的数据持久性。由于消除了同步持久化的开销，因此相对于基于共享存储的主备机复制方法，该方法可在万兆以太网环境中保证交易系统可用性的同时大大降低系统延迟。

## 2 基于 Paxos 算法的内存数据复制框架

图 1 所示为我们设计的一种基于 Paxos 算法的内存数据复制框架。该框架在客户端-服务器模式的基础上，通过对服务器进行多结点冗余备份来实现高可用性。服务器端的业务数据驻留于内存数据区以提高访问速度，而客户端通过发送更新操作来对服务器上的业务数据进行访问与更新。

在各服务器结点中，有唯一的主结点，其余为从结点。主、从结点维护相同的内存数据，通过状态机复制的方式来实现数据同步，即通过保证各结点上所接收更新操作的顺序一致，且更新操作在内存数据上具有确定性执行结果，来保证在各结点上取得相同的内存数据状态。

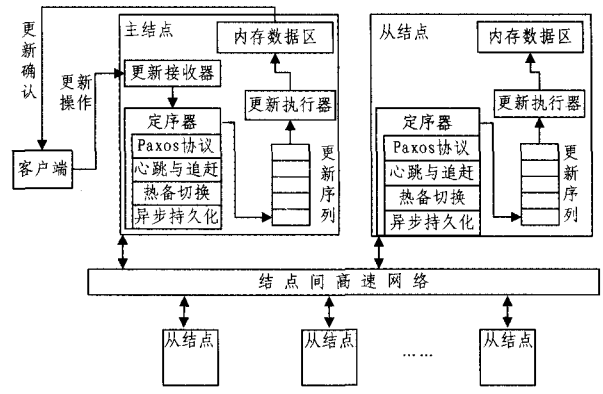


图 1 基于 Paxos 算法的内存数据复制框架

为保证各结点更新顺序一致，引入定序器模块。该模块通过在各结点间的高速网络上执行 Multi-Paxos 协议，将主结点所接收的更新操作在各结点之间进行全局定序与复制，同时提供后文介绍的与协议密切相关的心跳消息与落后追赶、主结点选举与热备切换、更新操作的异步持久化等机制实现。定序器在少于半数的结点发生故障的情况下，仍然能够保证各结点间数据的一致性，并持续地提供服务。

## 3 证券交易系统内存数据复制架构

本节中将结合证券交易系统场景，分析上节所述基于 Paxos 算法的内存数据复制框架的具体实现与应用。

首先，对于本文所讨论的证券交易系统场景进行限定。由于证券交易系统使用专用网络，恶性故障极少发生，因此在本文的容错设计中暂仅考虑非拜占庭故障[14,15]。另外，由于撮合引擎为证券交易系统的核心组件，且内存数据的复制在撮合主机与备机间完成，因此将本文系统设计范围限定为撮合系统。其输入和输出分别是订单消息和成交确认消息。实际系统中，订单路由为撮合系统提供输入并接收输出。

对应于上节的内存数据复制框架，撮合主机为框架中的服务器结点，其内存数据区中维护着订单簿等与交易相关的数据结构，而待撮合的证券订单即可视为作用在内存数据区之上的更新操作，订单撮合即为更新操作的执行。

图 2 所示为基于 Paxos 算法的撮合系统的基本架构，订单路由器作为客户端，持续向撮合系统中发送订单。由撮合主机接收订单输入并发起定序复制过程，将订单复制到各备机，在完成定序之后便可将订单交由撮合器进行撮合，撮合完成后主机将返回订单确认。

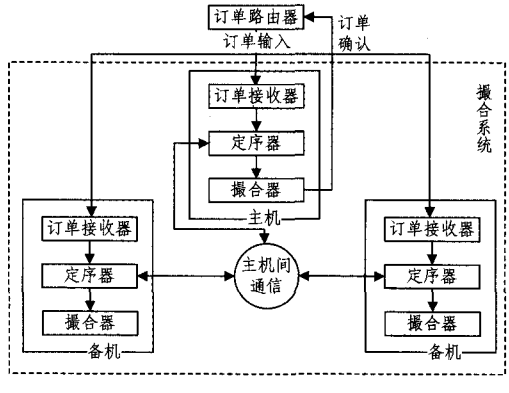


图 2 基于 Paxos 算法的证券撮合系统基本架构

撮合系统中各结点都包含 3 个主要模块，即订单接收器、定序器和撮合器。其中，订单接收器可靠、有序地接收订单路

由器发送的订单；定序器通过 Multi-Paxos 协议与其它定序器协商确定订单处理的全局顺序，实现订单数据复制；而撮合器对订单进行撮合，主机撮合成交后需向订单路由器返回确认消息。

各结点在内存中维护订单序列。经定序器定序后，各结点订单序列保持一致，且各结点撮合器按照相同规则撮合订单，因此撮合达到一致的内存数据状态。若主机发生故障，完全复制了主机内存数据状态的备机可以接替主机，完成热备切换。

因职能不同，主、备机的订单处理过程差异较大。图 3 所示为主机的订单处理流程，从订单接收到确认发送，它体现了订单处理的完整过程。由于订单路由器可能因确认消息超时而重发订单，系统需检查订单“是否已处理”以避免重复处理。检查可通过在订单中增加首次发送时间戳，并将其与主机上最新定序订单时间戳比较来实现。

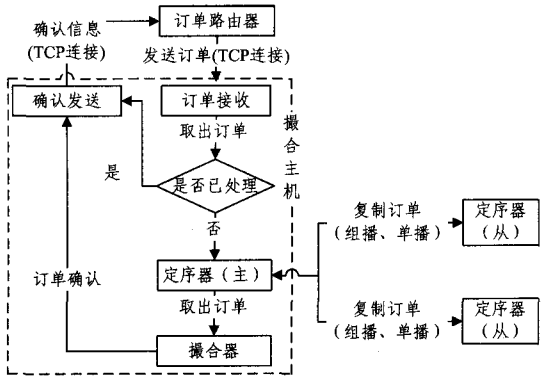


图 3　主机订单处理过程

图 4 所示为备机的订单处理流程，体现了备机的两个职能：(1)订单路由可能将订单发送到备机，备机收到订单后不做处理，仅返回主机地址信息，订单路由器据此信息将订单重发给主机；(2)备机定序器（以下称从定序器）在主机定序器（以下称主定序器）的协调下完成订单的定序与复制，然后将定序后的订单交给撮合器以实现内存数据状态的同步，但不发送确认。定序器与撮合器之间增加缓存队列是为了实现定序和撮合的异步执行，以避免因备机的撮合过程阻塞整个订单的处理过程。

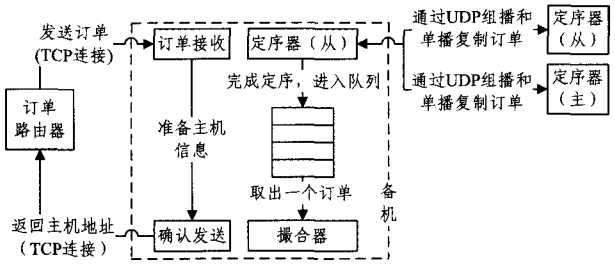


图 4　备机订单处理过程

上述分析表明，框架中的定序器封装了各结点通过 Multi-Paxos 协议对订单进行全局定序和复制的功能实现，是系统实现高可用性与低延迟的关键。

## 4　定序器算法实现方法

定序器是以上框架中最为关键的模块，为此我们设计了一系列算法实现。本节中将首先说明 Multi-Paxos 协议在定序器中的部署实现，之后分析故障场景下新主定序器选举与热备切换，最后引入落后结点的进度追赶和故障结点的恢复机制。

### 4.1　Multi-Paxos 协议部署

为实现容错复制，系统结点间运行 Multi-Paxos 协议。协议中的参与者可分为 proposer，acceptor 和 learner 3 种角色。其中，proposer 发送待复制的提议值并发起对该值的投票请求，acceptor 对提议值进行投票，learner 对最终投票通过的决议值进行学习并执行相应处理操作。此外，协议中还需有向 proposer 发送待复制更新操作的 client。

为高效运行 Multi-Paxos 协议，并减少不同角色间所需发送的消息数，各结点的定序器都兼任协议中的多重角色。表 1 中给出了 Multi-Paxos 协议中的角色与系统模块的对应关系，各角色职能将在后续的算法描述中予以体现。

表 1　Multi-Paxos 角色在撮合系统中的对应模块

| 角色 | 对应模块 |
|---|---|
| client | 订单接收器 |
| proposer | 各结点的定序器都具有 proposer 职能。只有主定序器的 proposer 职能处于激活状态，充任 leader 角色。 |
| acceptor | 各结点的定序器 |
| learner | 各结点的定序器 |

系统中，待定序与复制的数据为订单数据，对应于协议中的提议值。每个经过全局定序的订单将被分配一个订单序号来标识其撮合顺序，该序号在协议中称为实例编号。Multi-Paxos 协议中引入了视图的概念，视图描述了一个主定序器从取得领导权到失去领导权的时间区间，其仅在主定序器切换时改变。视图编号唯一标识了一个视图，等同于基本 Paxos 协议中的投票发起权编号(ballot number)。视图编号在系统启动后从 0 开始，随主结点变迁而递增取值。

基本 Paxos 协议将每个通信轮次分为 prepare 与 accept 两阶段。我们在其 Multi-Paxos 部署中可将订单定序过程分为主定序器切换状态与主定序器稳定状态两种情况，其执行不同协议过程。通过对 Paxos 协议进行扩展，得到如下协议：

1)主定序器切换情况下，新的主定序器首先发起 prepare 阶段以获取各定序器上已有订单信息，使其达到最新数据状态，并发起必要的 accept 阶段来处理已发送但未完成定序的订单。

算法 1 中给出了在此情况下新的主定序器所执行的协议，在执行完该协议过程后即进入主定序器稳定状态，并开始处理来自订单接收器的新订单。算法 2 给出了每个定序器（含主定序器）处理 prepare 请求、accept 请求以及 accept 响应的协议过程伪码。对算法伪码中出现的变量说明如下：

pre_req 与 pre_res：表示 prepare 请求和响应。在响应中，Set_seq 表示实例编号大于 $S$ 且已定序的订单信息集合，Set_unseq 表示实例编号大于 $S$ 且未定序的订单信息集合。上述两集合分别包含形如$(S,O)$和$(S,V,O)$的元组，其中 $S$，$V$，$O$ 分别表示实例编号、视图编号与订单值。

acc_req 与 acc_res：表示 accept 请求和响应。

local_view_no：本地视图编号。

local_max_seq_no：本地已定序到的最大实例编号，其含义是小于等于该编号的实例均已定序。

SequencedO、ProposedO 与 ProposedV：已定序订单值、已提议的订单值及其视图编号。3 者需以对应实例编号为索引进行访问，且初始值为 NULL。

$N$:系统中结点个数。

**算法 1** 新主定序器当选后的协议过程

```
1. After being elected as a new master
2.   V=local_view_no
3.   S=local_max_seq_no
4.   send pre_req(V,S) to all sequencers
5.   wait until more than N/2 pre_res are received
6.   for each pre_res(V,Set_seq,Set_unseq)
7.     for each (S′,O′) tuple in Set_seq
8.       SequencedO[S′]=O′
9.       check and update local_max_seq_no
10.    end for
11.    for each (S″,V″,O″) tuple in Set_unseq′
12.      if SequencedO[S″]==NULL and ( ProposedV[S″]==
           NULL or ProposedV[S″] < V″ )
13.        ProposedV[S″]=V″
14.        ProposedO[S″]=O″
15.      end if
16.    end for
17.  end for
18.  for each S‴ ever proposed or sequenced
19.    if SequencedO[S‴]==NULL and ProposedO[S‴] != NULL
20.      send acc_req(V,S‴,ProposedO[S‴]) to all sequencers
21.      wait until more than N/2 acc_res(V,S‴) are received
22.      SequencedO[S‴]=ProposedO[S‴]
23.      update local_max_seq_no if necessary
24.  end if
25.  end for
```

**算法 2** 定序器对 3 种协议消息的处理

```
1. After receiving pre_req(V,S)
2.   if V >=local_view_no
3.     local_view_no=V
4.     Set_seq=∅
5.     Set_unseq=∅
6.     for each seq sequenced or proposed with seq > S
7.       if SequencedO[seq] != NULL
8.         Set_seq=Set_seq∪{(seq,SequencedO[seq])}
9.       else if ProposedOrder [seq] != NULL
10.          Set _unseq=Set_seq∪{(seq, ProposedV[seq], Pro-
               posedO[seq])}
11.      end if
12.    end for
13.    send pre_res(V,Set_seq,Set_unseq) to master sequencer
14.  end if
15.
16. After receiving acc_req(V,S,O)
17.  if V >=local_view_no
18.    local_view_no=V
19.    ProposedV[S]=V
20.    ProposedO[S]=O
21.    send acc_res(V,S) to all sequencers
22.  end if
23.
24. After receiving acc_res(V,S)
25.  if V >=local_view_no
26.    local_view_no=V
27.    if more than N/2 acc_res(V,S) are received
28.      SequencedO[S]=ProposedO[S]
29.      update local_max_seq_no if necessary
30.    end if
31.  end if
```

2)主定序器稳定的情况下,在接收到订单后,主定序器将跳过 prepare 阶段,直接发起 accept 阶段对订单进行定序复制,算法 3 给出了该过程的伪码。各定序器仍然按照算法 2 对相应协议消息进行处理。

**算法 3** 主定序器稳定状态下,主定序器对新订单的处理协议

```
1. After receiving a new order O
2.   V=local_view_no
3.   S=local_max_seq_no+1
4.   send acc_req(V,S,O) to all sequencers
5.   wait until more than N/2 acc_res are received
6.   SequencedO[S]=ProposedO[S]
7.   local_max_seq_no=local_max_seq_no+1
```

### 4.2 主结点选举与热备切换

各定序器周期性地向其他定序器发送心跳消息以通告自身的存在。主定序器发生故障后,各定序器将检测到其故障,并陆续向其它结点发送视图变更消息,开始选举新主定序器的过程。

我们通过视图编号来约定主定序器切换的顺序:系统中的结点数为 $N$,系统启动时为每个结点指定不同的 *node_id*,取值从 0 至 $N-1$,而每个结点都被分配到视图编号集合$\{v|v\%N=node_id$ 且 $v\geqslant 0\}$。当视图编号 $v$ 的主结点发生故障后,将首先尝试选举 *node_id* 为$(v+1)\%N$ 的结点为主结点,若该结点也发生故障,则尝试选举 *node_id* 为$(v+2)\%N$ 的结点,依此类推。

算法 4 即为主结点选举协议。在检测到主结点故障后,结点将视图变更消息发送给其它结点,并在该消息中尝试使用下一视图编号。当一个结点收到含有比本地视图编号大的视图编号的消息时,它将记录并接受该编号,并发送视图编号,继续将其转发给其它结点。

**算法 4** 主结点选举协议

```
1. Try_view_no(V)
2.   tried_view=V
3.   send view_change(tried_view) to all sequencers
4.   wait until more than N/2 view_change(tried_view) are received
5.   if waiting timeout
6.     Try_view_no(V+1)
7.   end if
8.
9. After detecting master failure
10.  if local_view_no is not locked
11.    Try_view_no (local_view_no+1)
12.  end if
13.
14. After receiving view_change(V)
15.  if detect master failure and local_view_no is not locked
16.    if V > tried_view
17.      record this view change
```

```
18.        Try_view_no(tried_view)
19.      else if V==tried_view
20.        record this view_change
21.        if more than N/2 view_change(tried_view) are received
22.          local_view_no=tried_view
23.          lock local_view_no for a while to finish master switching
24.        end if
25.      end if
26.    end if
```

各结点通过视图编号本身约定的选举顺序来初步协定视图编号，之后暂时锁定该视图编号，直至对应的主结点完成切换状态，进入稳定状态。协议中初步协定了某视图编号后，给予结点足够长的时间完成整个选举协议，超时后才允许尝试更大视图编号，同时各结点只有在检测到主结点故障时才尝试更改视图编号，以促进选举的完成。此外，在心跳消息中广播结点自身已确定的视图编号可使其它结点尽快确定其视图编号，从而加速选举。

新的主定序器需执行 4.1 节介绍的主定序器切换情况下的协议，新主机通过学习与撮合使自己达到最新内存数据状态，完成旧主机未完成的定序工作，之后执行主定序器稳定情况下的协议，正常进行定序与撮合，从而完成热备切换。

### 4.3 进度追赶与故障恢复

Paxos 算法只需有超过半数结点返回响应即可使协议取得进展，因而可能产生进展落后的结点。系统中需为落后结点提供追赶机制，确保系统持续、稳定地取得进展。解决方案是在心跳消息中包含结点进展信息，这些信息包括已定序的最新订单序号、已撮合的最新订单序号、本地最新视图编号等。各结点收到心跳消息后与本地状态比对，若发现本结点落后较多，则向该结点发送专门的追赶消息，请求本地缺少的订单，以追上系统进度。

系统中故障结点需要恢复并重新加入系统，否则正常结点将持续减少直至不足半数而使系统阻塞。在前文所述的分布式存储系统中，Paxos 算法会在发送协议消息前将自身协议状态记录到磁盘上，从而使得故障结点只需在重启后从磁盘中恢复出状态即可继续运行，与暂时网络中断的正常结点没有区别。但在低延迟内存数据复制中，不容许协议过程中频繁地对协议状态进行持久化，需消除持久化并保证内存数据状态全部丢失的情况下，恢复后的结点仍不会违背先前协议中的承诺，以保证数据一致性。

由于提供了追赶机制并假设了同时发生故障的结点不会超过半数，因此消除协议持久化的结点恢复方法如下：

(1)对于通过追赶机制从其它结点上学习到的已定序的实例信息，任何时间都可以对其协议请求进行响应，因为该实例对应订单已完成定序，响应不会造成数据的不一致。

(2)对于未完成定序的实例，并不能马上进行响应，因为该结点可能在故障前对其做出了一定承诺，但该信息因结点故障已失去，所以为了不违背可能已经做出的承诺，该结点需跳过该实例的协议。因此，刚恢复的结点需在目击了一个完整的协议实例过程之后，才能重新加入到后续实例的协议中。然而，网络的异步性使得所目击的协议实例并不一定完全开始于结点重启后，有两种方法可对该问题进行补救：1)在实际系统中，一条消息在网络中存在的最长时间有限，因此在等待足够长的时间后能保证看到的新协议消息是发生在结点重启后的；2)通过增加要求目击的完整实例数目，可以进一步确保不会违背故障前对其的承诺。更严格的故障恢复实现需要各结点的时钟同步机制，以确保结点所参与的协议实例开始于结点重启之后，本文不对其详细讨论。

对于可能发生的超过半数的结点同时故障的情形，在各结点故障发生时机并不集中的前提下，一种可能的处理方法是对系统结点状态进行实时监测。在有较多结点发生故障的情况下，将系统中的各结点切换到对协议状态进行持久化的协议版本，直到系统中未故障结点数恢复到较为安全的数目才切换回不进行协议状态持久化的协议版本。

## 5 异步持久化与内存数据快照

基于共享存储的主备机复制采取批量持久化来提高系统的吞吐量，但订单延迟仍然受限于磁盘单次写操作的延迟。而本文采用对订单进行异步持久化的方法，使订单在定序并撮合后即可直接返回确认，同时可以利用磁盘持久化的高吞吐量来保证较强的数据持久性。

我们通过一个循环页表的设计来实现订单的异步持久化，并减少对于订单处理主流程的影响。如图 5 所示，将内存中已定序的订单数据组织为一系列数据页，每个数据页中保存若干条订单。将完成定序后的订单加入到页表队尾的未满数据页中。算法 5 与算法 6 分别给出了向页表加入订单以及对页表订单进行持久化的伪码。其中，pages 为页表数据页，head 与 tail 分别为页表的队首与队尾数据页的下标索引，而 page_num 表示页表中的数据页总数。

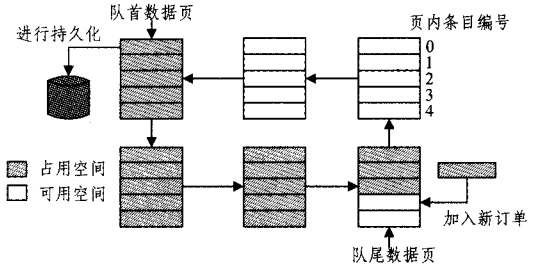


图 5 用于异步持久化的循环页表设计

**算法 5** 向队尾数据页加入订单

```
1. After receiving a new order O
2.   while true
3.     if pages[tail] is not full
4.       append O to pages[tail]
5.       return
6.     else if (tail+1) % page_num ! =head
7.       tail=(tail+1) % page_num
8.       append O to pages[tail]
9.       return
10.    end if
11.  end while
```

**算法 6** 队首数据页持久化

```
1. while true
2.   if pages[head] is full
3.     write pages[head] to disk
4.     head=(head+1) % page_num
5.   end if
6. end while
```

向队尾数据页中添加新订单与对队首数据页进行持久化

并不冲突，可以并发执行。以数据页为单位进行持久化，平摊了多条订单进行持久化的开销。数据页的大小需根据订单复制的吞吐量与数据页持久化的吞吐量来确定，使两者相互匹配，令循环页表能被无阻塞地循环使用。

订单撮合耗时较多，若仅对订单进行持久化，则在结点故障重启后需经过较长的重新撮合过程才能达到最新状态。通过创建内存数据快照来解决此问题，即将较近时刻的内存数据区进行持久化。结点恢复时可以先将快照加载到内存数据区中，再学习和撮合快照创建时间点之后的订单以达到最新数据状态。创建快照期间需锁定内存数据区，为减少对系统性能的影响，通过预先指定部分结点定期进行内存数据快照创建工作，在快照保存后再进行追赶。

## 6 实验

实验使用 6 台主机，其中 1 台作为订单路由器，5 台作为撮合主机。各主机配置均为 16GB 内存，四核 CPU 且主频为 2.4GHz，主机间通过万兆以太网连接。使用 C 语言实现上述设计的原型，操作系统为 SUSE Linux。原型中暂将订单撮合过程忽略，仅考察订单复制性能。通信方面，订单路由器与撮合主机间通过 TCP 连接发送订单和接收确认；各撮合主机通过 UDP 多播和单播进行通信。

实验中在持久化时关闭写缓存以保证订单数据及时刷新至磁盘。实验所使用的磁盘单次写操作用时随着所写数据大小的变化如图 6 所示。

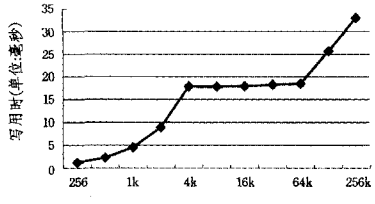


图 6 单次磁盘写操作用时

证券交易系统中订单的典型大小约为 256 字节，图 6 中写 256 字节用时在 1 毫秒以上。若进行同步持久化，则订单延迟至少为毫秒级。而异步持久化可平摊多条订单的持久化，并且不阻塞新订单复制过程。

基于消息传递的复制中可将多条订单合并进行批量复制，以平摊通讯开销，提高吞吐量。原型中结点间通过 UDP 组播通讯，受限于 UDP 的消息尺寸，因此将一次所能发送的最大消息大小限定为 4kB。5 个结点对订单进行复制并做异步持久化，不同消息大小下，订单路由器与主结点上的复制延迟如图 7 所示。

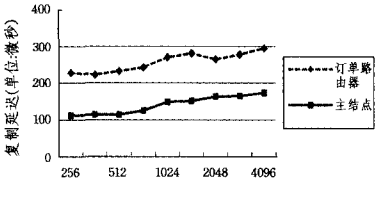


图 7 消息大小对复制延迟的影响

结果表明，在万兆以太网环境下，随着消息大小的增加，复制延迟呈现非常平缓的增长。主结点延迟保持在 100 至 200 微秒之间，而订单路由器延迟保持在 200 至 350 微秒之间。因此，在网络稳定的情况下，订单复制延迟可保证在几百微秒量级。网络复制的吞吐量可达每秒几十兆字节，与磁盘吞吐量相匹配，因此异步持久化不会阻塞复制过程。

实际系统中，基于共享存储的主备机复制通常采用企业级存储区域网络[16]，其单次写操作一般在毫秒量级，日志文件的事务保护进一步增加了延迟。相比共享存储复制，本文的内存数据复制方法在性能方面优势明显。

在运行过程中关闭主结点以测试热备切换的正确性，发现系统可在数秒内检测到故障并选举出新主结点，由新主结点进一步完成热备切换。由于实验中仅对订单定序而不撮合，因此主结点追赶进度用时很少，通常在一分钟以内。

**结束语** 针对基于共享存储的主备复制技术存在同步持久化的性能瓶颈而难于大幅改进延迟性能的问题，本文提出了作为替代的基于消息传递的内存数据复制方法。该方法能够充分利用万兆以太网环境通过消息传递方式复制订单数据，并通过 Paxos 协议来保证良性故障下数据的一致性。方法中针对内存复制对 Paxos 协议作了消除持久化的优化，并对故障结点恢复等问题提出了相应的解决方法。实验验证了该方法在万兆以太网环境下将系统延迟降至百微秒量级的可行性。未来工作中，我们将考察不同网络环境与协议、不同结点数对协议性能的影响，对其稳定性做进一步评估；并在此基础上研究如何对系统吞吐量进行优化，以及如何将协议扩展到拜占庭故障环境。

## 参 考 文 献


[1] Stavros H, Abadi D J, Samuel M, et al. OLTP through the looking glass, and what we found there[C]// Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. Vancouver, BC, Canada, 2008: 981-992

[2] Michael S, Samuel M, Abadi D J, et al. The end of an architectural era (It's time for a complete rewrite)[C]// Proceedings of the 33rd International Conference on Very Large Data Bases. Vienna, Austria, 2007: 1150-1160

[3] 武剑锋. 交易系统：更新与跨越[M]. 上海：上海人民出版社，2011

[4] 黄寅飞，王泊，武剑锋，等. 证券交易系统中的日志复制[J]. 计算机应用与软件，2011，28(9)：171-173

[5] 黄寅飞，黄俊杰，王泊，等. 证券交易系统中的事务恢复方法[J]. 计算机工程，2010，36(24)：241-243

[6] Schneider F B. Implementing fault-tolerant services using the state machine approach: a tutorial[J]. ACM Computing Surveys, 1990, 22(4): 299-319

[7] Leslie L. The part-time parliament[J]. The Distributed ACM Transactions on Computer Systems, 1998, 16(2): 133-169

[8] Leslie L. Paxos made simple [J]. ACM SIGACT News, 2001, 32(4): 18-25

[9] Mike B. The Chubby lock service for loosely-coupled distributed systems[C]// Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation. Seattle, USA, 2006: 335-250

[10] Tushar C, Robert G, Joshua R. Paxos made live-an engineering perspective[C]// Proceedings of the 26th annual ACM symposium on Principles of distributed computing. New York, USA, 2007: 398-407

[11] Marton T, Attila G, Holger R. PaxosLease: Diskless Paxos for Leases[R/OL]. http://scalien.com/pdf/PaxosLease.pdf, 2011-11-15


(下转第 166 页)

表 7　由 $Sob$ 生成的形式背景 $K_2$

| | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | 1 | | | | 1 | | 1 | | 1 | 1 | | 1 | | 1 | | 1 | | 1 | | | 1 | | | 1 | | 1 | | | 1 | 1 | | | | 1 | | 1 |
| 2 | 1 | | | 1 | 1 | | | 1 | | | | | 1 | | | | | | | | | | | | | | | | | | 1 | | | | | | |
| 3 | | | | | | | | | 1 | | | 1 | | 1 | | | | | | | | | | | | | 1 | | | 1 | | 1 | 1 | 1 | | 1 | | |
| 4 | | | | | | | | | | | | | | | 1 | 1 | | | | | | | | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | | | | 1 | | 1 | | | | | | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | 1 | | 1 | | | | | | | 1 | | 1 | | | | | 1 | | | 1 | | | | | | | |
| 7 | | 1 | | 1 | | 1 | | | | | | | | | | | | | | | | 1 | | | 1 | | | | | | | | | | | | |
| 8 | | | | | | | | | | 1 | | | | | | | | | | | | | | | | | | | 1 | | | | | | | | | |

(3)分层提取角色

对 $K_2$，根据概念格构造算法[12]得到概念格，如图 6 所示。

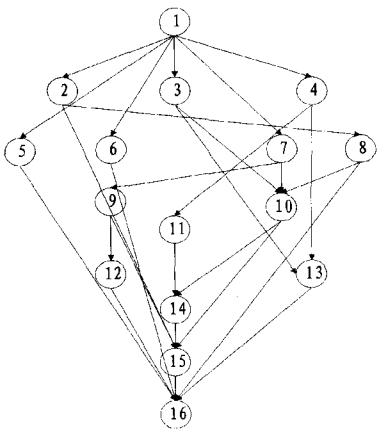


图 6　概念格 $CS(K_2)$的 Hasse 图

图 6 中每个形式概念的编号及外延和内涵分别为：1(12345678,∅)，2(14,$p$)，3(123,$F$)，4(27,$d$)，5(5,$qs$)，6(8,$jC$)，7(136,$ln$)，8(4,$op$)，9(16,$lnBE$)，10(13,$ilnF$)，11(7,$bdfvy$)，12(6,$lnuxBE$)，13(3,$ilnADFGHJ$)，14(2,$adehmF$)，15(1,$cgiklnprtwzBEFIK$)，16($\emptyset$,$a-K$)。按照形式概念的外延大小，可以将 $CS(K_2)$分为 3 个集合 $S_1$、$S_2$ 和 $S_3$，$S_1=\{5,6,8,11,12,13,14,15\}$，$S_2=\{2,3,4,7,9,10\}$，$S_3=\{1,16\}$。易知，$CS(K_2)=S_1\cup S_2\cup S_3$，其中 $S_1$ 表示基本角色集合，由 $Sob$ 可以直接得到；$S_2$ 表示泛化角色集合，通过概念格中的形式概念转换得到；$S_3$ 表示冗余角色集合。

由以上分析可知，这种方式可以使用概念格模型对需求信息进行分析并提取出系统中的角色及角色间的层次关系，能够与 CIS 中的需求紧密结合，降低系统开发成本。

**结束语**　本文在基于主谓宾方法获取需求与表示的基础上，引用形式概念分析中的概念格模型，将以主谓宾形式表示的与权限相关的需求转换为形式背景，然后构造出概念格，从概念格中提取出具有层次关系的角色及权限。通过实验可以看出，概念格模型在角色提取中是有效的，提取角色方式简单、直观。

本文实验中的角色依靠部门属性作为角色名称，这不够全面，在有些场合不适用。进一步的研究工作将考虑把角色作为一种普通的属性利用主语的各种属性构造概念格，并与权限概念格之间建立映射，找出影响主语的权限的关键属性集合。同时，考虑引入信息粒度度量方式，对提取出的角色进行评估。

## 参 考 文 献


[1] Zhang D, Ramamohanrao K, Ebringer T. Role Engineering using Graph Optimisation[C]// Symposium on Access Control Models and Technologies (SACMAT). 2007: 139-144

[2] Schlegelmilch J, Steens U. Role mining with orca[C]// Symposium on Access Control Models and Technologies (SACMAT). ACM, June 2005

[3] Molloy I, Li N, Li T, et al. Evaluating role mining algorithms[C]// Proceedings of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT). 2009: 95-104

[4] Ma Xiao-pu, Li Rui-xuan, Lu Zheng-ding. Role Mining Based on Weights[C]// SACMAT'10. Pittsburgh, Pennsylvania, USA, June 2010: 65-74

[5] 姚莉，张维明，王长缨，等. 基于多智能体的复杂信息系统开发方法研究[J]. 管理科学学报，2002，5(5)：44-54

[6] 王琨，袁峰，周利华. 复杂信息系统模型研究[J]. 计算机科学，2006，33(10)：119-123

[7] Focardi R, Gorrieri R, et al. Access Control: Policies, Models, and Mechanisms[C]// FOSAD 2000, LNCS 2171. 2001: 137-196

[8] Sandhu R, Coyne EJ, Feinstein H L, et al. Role based access control models[J]. IEEE Computer, 1996, 29(2): 38-47

[9] Zheng Yun-xiang, Wan Hai, Li Lei, et al. A new software requirement method based on subject, predicate and object Logic [C]// Software Process Workshop 2005 (SPW2005). Beijing, China, May 2005: 25-27

[10] 何云强，李建凤. RBAC 中基于概念格的权限管理研究[J]. 河南大学学报：自然科学版，2011，41(3)：308-311

[11] Ganter B, Wille R. Formal concept analysis: Mathematical foundations[M]. Berlin: Springer-Verlag, 1999

[12] 沈夏炯. 概念格同构生成方法研究及 IsoFCA 系统实现[D]. 上海：上海大学，2006


(上接第 144 页)


[12] Jason B, Chris B, Corbett J C, et al. Megastore: Providing Scalable, Highly Available Storage for Interactive Services[C]// Proceedings of the 5th Biennial Conference on Innovative Data Systems Research. Asilomar, California, USA, 2011: 223-234

[13] Marton T, Attila G. Keyspace: A consistently replicated, highly-available key-value store[R/OL]. http://scalien.com/pdf/Keyspace.pdf, 2011-11-15

[14] Leslie L, Robert S, Marshall P. The Byzantine generals problem [J]. ACM Transactions on Programming Languages and Systems, 1982, 4(3): 382-401

[15] 杨磊，黄浩，李仁发，等. P2P 存储系统拜占庭容错机制研究[J]. 计算机应用研究，2009，26(1)：4-8

[16] 苟彦，李华. 存储区域网络在联机事务处理下的性能研究[J]. 计算机科学，2011，38(Z10)：172-174