# FaRM: Fast Remote Memory快速远程内存

## Abstract

我们描述了FaRM的设计和实现，FaRM是一种新的主​​内存分布式计算平台，该平台利用了RDMA，相对于使用TCP / IP的现有主内存系统水平，将等待时间和吞吐量提高了一个数量级。 **FaRM将群集中机器的内存公开为共享地址空间。应用程序可以使用事务来透明地分配、读取、写入和释放地址空间中的对象。**我们希望这个简单的编程模型足以满足大多数应用程序代码的要求。 FaRM提供了两种机制，可在需要时提高性能：**通过RDMA进行的无锁读取**，以及支持**并置对象**和**函数传送**以启用有效的单机事务处理。 FaRM使用RDMA既可以直接访问共享地址空间中的数据，也可以进行快速消息传递，并且已经过精心调优，以实现最佳的RDMA性能。我们使用FaRM建立了类似于Facebook的键值存储和图形存储。它们都表现良好，例如，一台由20台计算机组成的集群可以每秒执行1.67亿次键值查找，延迟时间为31μs。

## Introduction

降低的DRAM价格已使构建具有数百GB DRAM的商用服务器具有成本效益。具有一百台计算机的集群可以容纳数十兆兆字节的主内存，足以存储许多应用程序的所有数据，或者至少缓存应用程序的工作集[11、38、39]。这有可能使应用程序执行小的随机数据访问，因为它消除了磁盘或闪存的开销，但是网络通信仍然是瓶颈。当系统继续使用传统的TCP / IP网络时，新兴的快速网络将无法解决此问题。例如，[16]中的结果显示了最新的键值存储执行[16]中的结果显示，尽管有大量请求批处理，但在使用TCP / IP的客户端服务器设置中，最先进的键值存储的性能比单机设置差7倍。

RDMA提供可靠的用户级别的远程内存读写。它实现了低延迟和高吞吐量，因为它绕过了内核，避免了复杂协议栈的开销，并且仅使用远程NIC进行远程内存访问，而无需使用远程CPU。 RDMA长期以来一直受到Infiniband的支持，但是由于Infiniband传统上价格昂贵并且与以太网不兼容，因此RDMA在数据中心中尚未得到广泛使用。如今，RoCE [27]通过具有竞争力的价格的数据中心桥接[25，26]支持以太网RDMA。

FaRM使用**RDMA写入**来实现快速消息传递原语，该原语相对于同一以太网上的TCP / IP，实现了消息速率和延迟方面的数量级改进。它还使用**单边RDMA读取**来实现对大多数工作量占主导地位的只读操作的双重两倍的改进[9，11]。我们没有开箱即用地提供这种性能（表面的性能）。通过仔细调整和更改操作系统和NIC驱动程序，我们将性能提高了八倍。

FaRM机器将数据存储在主存储器中，并且还执行应用程序线程。这样就可以进行**位置感知优化**，这很重要，因为访问本地内存的速度仍比RDMA快23倍。

FaRM将群集中所有计算机的内存公开为共享地址空间。线程可以使用具有严格可序列化性的ACID事务来分配、读取、写入和释放地址空间中的对象，而不必担心对象的位置。 FaRM提供了此简单编程模型的有效实现，该模型为大多数应用程序代码提供了足够的性能。**事务使用具有优化的两阶段提交协议的乐观并发控制**，该协议利用了RDMA。 FaRM通过使用**复制日志记录[39]到SSD**来实现可用性和耐用性，但也可以将其（SSD）部署为缓存cache[11，38]。

FaRM提供了两种机制来提高性能，而只需要对代码进行局部更改即可：**无锁读取**可以通过单个RDMA执行，并且可以严格地与事务进行序列化；并且**支持并行对象和函数传送**，以允许应用程序将分布式事务替换为优化的单机事务。

我们在FaRM之上设计并实现了一种**新的哈希表算法**，该算法结合了带有链接的跳式（hopscotch）哈希[21]和关联性，以实现高空间效率，同时需要少量的RDMA读取来进行查找：小对象读取仅需要1.04 个RDMA读取，占比为90％。我们利用FaRM支持**并置相关对象和运输事务**的方式来优化插入、更新和删除操作。（关于读写的优化）

我们使用YCSB [15]来评估FaRM哈希表的性能。我们将FaRM与使用TCP / IP进行消息传递的基准系统进行比较，并且性能优于MemC3 [16]（这是文献中最好的主存储器键值存储）。我们对由40 Gbps以太网连接的20台服务器的群集进行的评估证明了其良好的可扩展性和性能：FaRM在广泛的设置范围内提供了比基准系统更好的**数量级**吞吐量和延迟。

我们还使用FaRM实现了Facebook的Tao图形存储[11]版本。与[11]中报道的数字相比，FaRM再一次实现了更高的吞吐量和延迟。

## RDMA的背景

RDMA请求通过可靠的连接（也称为*队列对*）发送，而网络故障则显示为终止的连接。请求直接发送到NIC，而不涉及内核，并且由远程NIC处理且不会中断CPU。必须先向NIC**注册内存区域**，然后才能将其用于远程访问。在注册期间，NIC驱动程序将**页面固定在物理内存**中，将虚拟页面到物理页面的**映射**存储在NIC的页面表中，并返回客户可以用来访问区域的*region capability*。当NIC收到RDMA请求时，它将获得目标区域的页表，**将目标偏移量和大小映射到相应的物理页中，并使用DMA访问内存**。许多NIC（包括我们正在使用的NIC）保证以递增的地址顺序执行RDMA写入（但不读取）。 DMA操作在我们的硬件平台上是缓存一致的。

NIC的页表和连接数据的**内存有限**。因此，许多NIC（包括我们的NIC）将此信息存储在系统内存中，并**将NIC内存用作高速缓存**。访问未缓存的信息需要发出DMA，以通过PCI总线从系统内存中获取信息。这是卸载技术的普遍局限，需要**仔细使用可用内存**才能获得良好的性能。

RDMA长期以来一直受到HPC社区广泛使用的Infiniband网络的支持。已经存在具有数千个节点和完整的二等分带宽的部署（例如，[45]）。如今，Infiniband已变得与以太网在价格上具有竞争力[37]，但以太网在数据中心仍很普遍。

RoCE（融合以太网上的RDMA）硬件支持具有数据中心桥接扩展功能的以太网RDMA，许多交换机已经提供了扩展功能。这些扩展添加了基于优先级的流控制[26]和拥塞通知[25]。它们消除了由于拥塞造成的损失，并允许将RDMA与其他流量分开。**硬件管理连接状态和确认**，因此无需像TCP这样的协议栈即可确保可靠的传递。

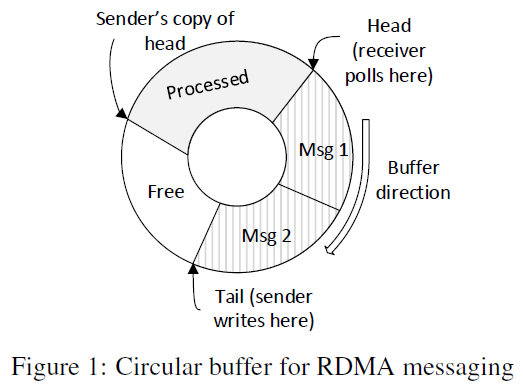
RoCE在机架级别具有价格竞争力：40 Gbps RoCE为$19 / Gbps，而10 Gbps以太网为$60 / Gbps，**但是RoCE的可扩展性令人担忧**。我们希望它可以扩展到数百个节点，并且正在进行改进以扩展到数千个节点的可扩展性的工作。本文介绍了使用40 Gbps RoCE在20机集群上的结果，但我们还在78个Infiniband集群上运行了FaRM。

## FaRM

本节介绍FaRM的设计和实现。首先讨论FaRM的通信原语，以及如何针对RDMA优化其实现。然后，它描述FaRM如何实现共享地址空间，以及如何确保以良好的性能对地址空间进行一致的访问。

### 3.1 通讯原语

FaRM使用**单边RDMA读取**直接访问数据，并使用**RDMA写入**来实现快速消息传递原语。如图1所示，该原语使用**循环缓冲区**来实现*单向*通道。缓冲区存储在接收方上，每个发送方/接收方对都有一个缓冲区。缓冲区的未使用部分（标记为“Processed”和“Free空闲”）保持为零，以使接收器能够检测到新消息。接收器会定期轮询在“Head”（头）位置的单词word，以检测新消息。头部中的任何非零值L表示**长度为L**的新消息。然后接收者轮询该消息的尾部trailer；当它变为非零值时，因为RDMA写操作以递增地址顺序执行，所以已接收到整个消息。消息被传递到应用程序层，一旦消息被处理，接收方将消息缓冲区**清零**并前进Head指针。



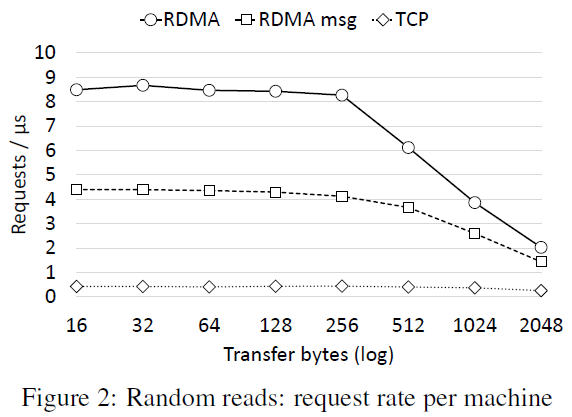
发送方使用RDMA将消息写入缓冲区尾部，并在每次发送时使尾部指针前进。

它维护接收者头部指针的本地副本，并且从不写超出该点的消息。

接收者通过使用RDMA将head指针的当前值写入发送者的副本中，从而**懒惰地**将处理后的空间提供给发送者。为了减少开销，接收者仅在处理了至少一半的缓冲区后才更新发送者的副本。发送者的头部副本始终落后于接收者的头部指针，因此可以保证发送者永远不会覆盖未处理的消息。

轮询开销随**通道数**线性增加，因此我们建立了从**每个线程**到**每个远程计算机**的单个通道。我们观察到78台机器的轮询开销可以忽略不计。我们还发现，在这种规模下，RDMA写入和轮询的性能明显优于更复杂的Infiniband发送和接收动词。在大型群集中，最好将RDMA立即写和共享接收队列一起使用[35]，这将使轮询开销保持恒定。

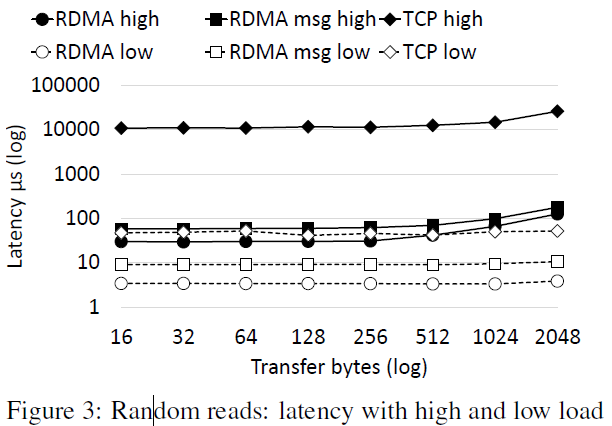
我们运行了一个微型基准测试，以比较FaRM的通信原语与TCP / IP在群集中的性能，该群集具有通过40 Gbps RoCE网络连接的20台计算机（更多详细信息，请参见第4节）。每台机器运行着多个线程，这些线程发出请求以一种all-to-all的通信模式从随机的远程机器读取随机的内存块。图2显示了针对峰值吞吐量进行了优化的配置中每台计算机的平均请求速率。在图的左侧，FaRM的通信原语在数据包速率上是瓶颈（可以发送更多的数据包），而在右侧则是比特率上（发送速率）的瓶颈。单边RDMA读取在2 KB请求大小时可实现近33 Gbps的比特率，而对于大于8 KB的请求大小，比特率​​可达到35 Gbps左右。



请求大小在16到512字节之间时（这是数据中心应用的典型值）（例如[9]），FaRM基于RDMA的消息传递比TCP / IP高11到9倍，而对于小于等于256字节的数据包大小，单边RDMA读取实现了2倍的额外改进，因为它们需要一半的网络数据包。我们预计，随着支持4倍消息速率的下一代NIC的出现，这种性能差距将进一步扩大[36]；单边RDMA读取不涉及远程CPU，而基于RDMA的消息传递将受CPU约束。

我们还测量了UDP吞吐量，发现它不到配置为最大吞吐量（启用Nagle）的TCP吞吐量的一半。因此，我们决定在本文的其余部分中将其与TCP进行比较。

图3显示了峰值请求率和仅使用配置为最小延迟的两台计算机的平均请求延迟。在所有请求大小中，峰值请求速率下的TCP / IP延迟比基于RDMA的消息传递的延迟至少高145倍。对于最大256字节的大小，使用**单边RDMA读取**可将等待时间减少两倍。**在无负载系统中**，在所有请求大小中，RDMA读取的延迟至少比TCP / IP低12倍，比基于RDMA的消息传递至少低3倍。微观基准表明，FaRM的通信原语可以同时实现低延迟和高消息速率。并且在所有请求大小上，均比基于RDMA的消息传递低3倍。微观基准表明，FaRM的通信原语可以同时实现低延迟和高消息速率。

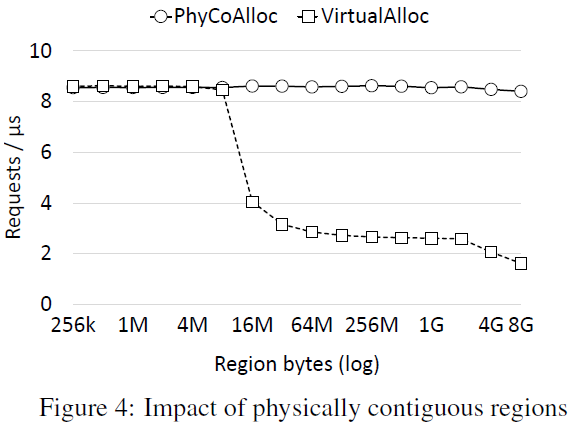


对于两个通信原语，要达到这样的性能水平并非易事，并且需要解决几个问题。

我们观察到的第一个问题是，随着我们增加了注册用于远程访问的内存量，RDMA操作的性能显着下降。原因是网卡空间不足，无法缓存所有页表。因此，它一直通过PCI总线从系统内存中获取页表条目。

我们使用较大的页面来减少NIC页面表中的**条目数**，从而解决了此问题。不幸的是，由于FaRM注册的大量内存，现有Windows和Linux中支持的大页面不足以消除所有获取操作。因此，我们实现了PhyCo，**这是一个内核驱动程序，在启动时会分配大量物理上连续且自然对齐的2 GB内存区域（2 GB是NIC支持的最大页面大小）**。 PhyCo将区域映射到在**2 GB边界对齐**的FaRM进程的虚拟地址空间。这使我们能够将NIC驱动程序修改为使用**2 GB的页面**，从而将每个区域的页面表条目的数量从超过一百万个减少到一个。

我们运行了随机读取基准，以比较使用VirtualAlloc和PhyCo分配区域时64字节RDMA读取的请求率。图4显示，使用VirtualAlloc向NIC注册超过16 MB的内存时，请求率下降了4倍;对于PhyCo，即使注册了100 GB的内存，请求率也保持不变。



当群集大小增加时，我们还观察到请求率显着下降，这是因为NIC用尽了空间来缓存队列对数据。在每对线程之间使用队列对需要每台机器2×m×t2个队列对（其中m是机器数，t是每台机器的线程数）。

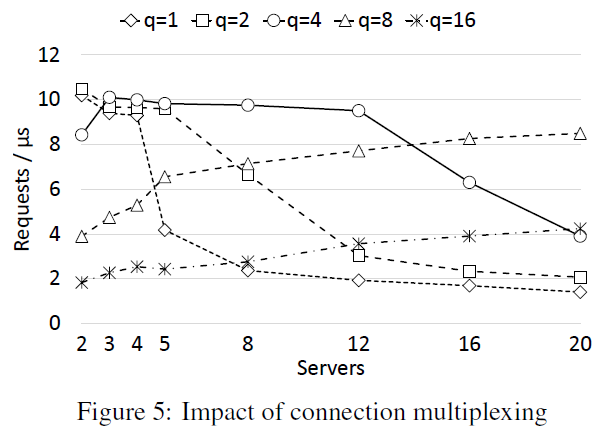
我们使用线程和每台远程计算机之间的单个连接将其减少为2×m×t。此外，我们以**NUMAaware方式引入了q个线程之间的队列对共享**，每台机器总共有2×m×t / q个队列对。这需要在并行性之间进行权衡，以减少NIC上的队列对数据量。

我们使用64字节传输运行随机读取基准，同时更改群集大小和q的值。

图5显示q的最佳值取决于群集的大小。较小的值可提供更多的并行性和较低的共享开销，从而在较小的群集中提供更好的性能，但是它们还需要更多的队列对数据，这会导致较大的群集降低性能。在本文的其余部分，我们将使用这些结果为每个聚类大小选择最佳的q值。我们希望将来通过使用**动态连接**传输[36]解决此问题，该技术通过按需设置连接来提高可伸缩性。

早期的实验表明，使用**中断和阻塞**可能会使RDMA延迟增加四倍。因此，我们使用基于**事件的编程模型**。每台FaRM机器都运行一个用户级进程，并将线程固定到硬件线程。

线程运行事件循环，该循环执行应用程序工作项并轮询基于RDMA的消息的到达和RDMA请求的完成。该轮询是在用户级别完成的，并且不涉及操作系统。



### 3.2 架构和编程模型

FaRM的体系结构是由上一节中介绍的性能结果驱动的。 FaRM的通信原语速度很快，但对主存储器的访问仍可达到高达23倍的请求率。因此，我们设计了FaRM，以通过在同一台计算机上**并置数据和计算**来使应用程序提高性能。 FaRM机器将数据存储在主存中，并且它们还执行应用程序线程。群集中所有计算机的内存均作为共享地址空间公开，可以使用**单边RDMA读取**该共享地址空间。当前，我们在整个集群中支持单个FaRM保护域（即单个内存池）。

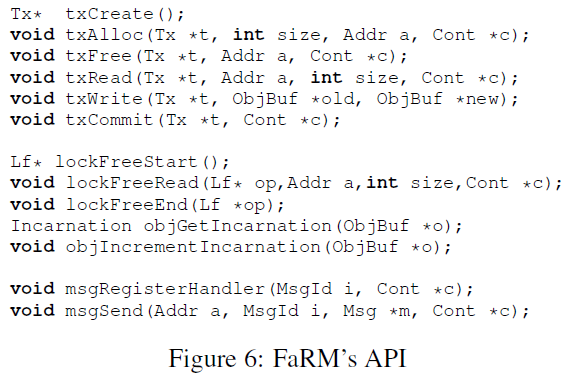


图6显示了FaRM接口中的主要操作。 FaRM提供了基于事件的编程模型。需要轮询才能完成的操作带有一个continuation参数，该参数由一个**continuation函数**和一个**上下文指针**组成。当操作完成时，将调用**continuation**函数，并向其传递操作的结果和上下文指针。**continuation总是在启动操作的线程上调用**。

FaRM提供**严格可序列化的ACID事务作为确保一致性**的通用机制。应用程序通过创建事务上下文来启动事务。他们可以在事务内部使用txAlloc和txFree分配和释放对象。

分配返回不透明的64位指针，**该指针可用于访问对象或存储在对象字段中以建立指针链接的数据结构。**应用程序可以通过将现有对象的地址提供给txAlloc来请求将**新对象分配给现有对象附近**。 FaRM尝试将两个对象存储在同一台计算机上，即使在从故障中恢复或添加新计算机之后，也将它们保留在同一台计算机上。这允许应用程序并置通常一起访问的数据。

给定对象的地址和大小，可以使用txRead操作读取对象。它分配一个对象缓冲区，并使用RDMA将对象的数据和元数据读入缓冲区。完成时，它将对象缓冲区传递给continuation。要更新对象，事务必须首先读取对象，然后调用txWrite创建对象缓冲区的可写副本。应用程序通过调用txCommit提交事务，txCommit返回结果并释放所有分配的缓冲区。**事务可能由于冲突或失败而中止；否则，将执行写操作**。

通用分布式事务提供了一个简单的编程模型，但是对于实现性能至关重要的操作而言可能过于昂贵。 FaRM的API使应用程序可以实施**有效的无锁只读操作**，这些（读）操作可随事务进行序列化。lockFreeStart和lockFreeEnd用于括起无锁操作。 lockFreeRead与txRead相似，但是它分配的任何对象缓冲区都由lockFreeEnd释放。 FaRM还公开了对象化身（副本？），可用于将多个无锁读取组合到更复杂的操作中。事务和无锁操作在第3.4节和第3.5节中进行了描述。

最后两个API操作用于将基于RDMA的消息发送到存储对象的计算机中的线程，**从而允许将事务传送到存储该对象的服务器**。再加上在同一台机器上并置相关数据的能力，这使得可以用单台机器事务来代替分布式事务，这大大降低了成本。

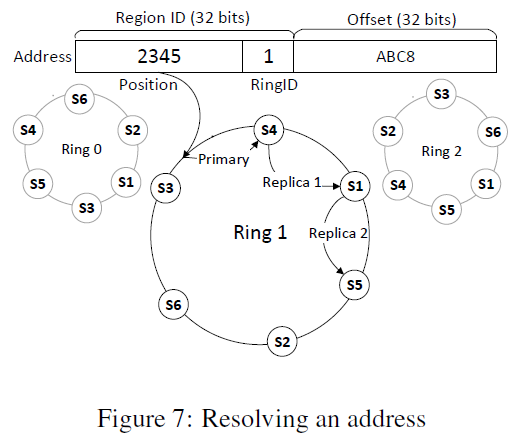
FaRM还提供分配、读取和释放对象数组的功能。这样可以通过单个RDMA有效读取数组中的连续元素。

FaRM使用复制日志记录以提供ACID事务，具有严格的可序列化性和高可用性，实现这些基于以下特性：崩溃故障、每个副本组最大故障数的限制、为安全起见在非同步系统中的时钟漂移的限制以及最终生命周期的同步。在本文中，我们没有描述或评估从故障中恢复的情况，但是，本文中报告的常见情况（非故障）性能包括复制和日志记录的所有开销。

我们使用此接口实现了分布式哈希表（第3.6节）和类似于Facebook的Tao [11]（第4.4节）的图形存储。

### 3.3 分布式内存管理

**FaRM的共享地址空间由许多2 GB的共享内存区域（regions）组成，这些区域是地址映射的单位、恢复的单位以及RDMA在NIC上的注册单位**。共享地址空间中对象的地址由32位**区域标识符**和相对于该区域开始的**32位偏移量**组成。为了访问对象，FaRM使用**一致性哈希**的形式[31]将区域标识符映射到存储对象的机器。如果该区域存储在本地，FaRM将获取该区域的基地base址并使用本地内存访问。否则，FaRM将与远程计算机联系以获得该区域的功能（描述符？），然后使用该功能，（获得）地址中的偏移量和对象大小来构建RDMA请求。远程区域的功能（描述符）将被缓存，以提高性能。



使用单跳分布式哈希表[6]来实现一致性哈希。将每台机器通过使用k个hash函数对它的IP地址进行散列，可以映射到k个虚拟环。 FaRM使用多个环以允许在RAMCloud中并行恢复多个区域[39]，并且还改善了负载平衡[44]。当前，我们使用k =100。32位共享区域标识符可同时标识一个环和环中的位置。然后，该区域的主版本和从副本将存储在环中位置随后的r台计算机上。图7显示了一个简单的示例，其中k = 3，r =3。给定集群中的一组机器，可以在本地执行区域到机器的映射。使用Zookeeper [24]可以可靠地维护集群成员资格。

内存分配器被组织为三级层次结构（平板slabs，块blocks和区域regions），以减少同步开销（如并行分配器[10]）。在最低级别上，线程具有专用的平板slabs分配器，该分配器从**大块（block）中分配小对象**。**每个块用于分配相同大小的对象**。FaRM支持256种不同的大小，从64字节到1 MB。选择大小后，平均碎片率为1.8％，最大值为3.6％。

**将以适合该对象的最小尺寸类别分配对象**。平板分配器在每个对象的**标头中**使用单个位来标记其已分配。当分配或释放对象的事务提交时，该状态被复制。在恢复过程中，该状态被扫描以重建分配器数据结构。

这些块是从**机器范围**的**块分配器**获得的，该分配器从共享内存区域分配块。**它将区域划分为大小为1 MB倍数的块**。每个区域都有一个表，（表中）每个块有一个8字节的分配状态。这些**区域**是从**集群范围**的区域分配器获得的。区域分配器使用PhyCo为该区域分配内存，然后向NIC注册该区域以允许远程访问（如第2节所述）。它通过**随机选择一个环**和确保**本地节点**存储**主副本**的环中位置来选择区域的标识符。有关区域和块分配的信息将在分配时复制。（确保数据尽量在本地申请的）

FaRM允许应用程序在分配对象时提供**位置提示**，该提示是现有对象的地址。FaRM尝试按以下顺序分配对象：**在与提示相同的块中、在同一区域中、或在同一虚拟环中具有附近位置的区域中（~~即相邻的虚拟全局地址~~）**。这样可确保即使在失败和重新配置之后，分配的对象和提示（地址）也仍然很可能同时并置（紧挨存储）在主对象和副本上。**如果提示是存储在另一台服务器上的地址，则使用RPC执行到远程服务器的分配。**

### 3.4 Transactions

FaRM支持将分布式事务作为确保一致性的通用机制。我们的实现使用乐观并发控制[32]和两阶段提交[18]来确保严格的可序列化性[41]。**事务上下文记录由事务读取的对象的版本号（读集），由事务写入的对象的版本号（写集），并缓冲写操作**。在提交时，运行事务的计算机充当**协调器**。首先向所有参与者发送准备prepare消息，这些消息是**写集**中对象的主副本和从副本。主副本将锁定修改后的对象，并且主副本和从副本都记录消息，然后将回复发送回去。收到所有参与者的答复后，协调器将验证validate消息发送到读取集中的主对象，以**检查事务读取的版本是否最新**。如果读取集验证成功，则协调器首先将提交消息发送给参与者从副本，然后再发送给参与者中的主副本。主副本更新已修改的对象并对其进行解锁，并且主副本和从副本均记录提交消息。如果任何修改的对象被锁定、读集验证失败或协调器未能收到所有准备和验证消息的答复，则事务中止。

FaRM副本将日志保留在SSD上。为了提高日志记录性能，他们使用了几兆字节的非易失性RAM [2]来保存循环消息缓冲区和缓冲日志条目[39]。当缓冲区已满时，将flush条目，并在日志半满时调用日志清理。这些日志用于实现类似于RAMCloud [39]的并行恢复机制。

**两阶段提交协议是使用基于RDMA的消息传递实现的**，该消息显示具有非常低的延迟。通过减少持有锁的时间，这可以减少冲突并提高性能。尽管进行了这些优化，但两阶段提交对于实现普通案例操作而言可能**过于昂贵**。

在常见情况下，FaRM提供两种机制来实现良好的性能：单机事务和无锁只读操作。

**应用程序可以通过在同一主副本和同一从副本上将事务访问的对象并置（紧挨放置），并将事务传送到主副本来使用单机事务。**在这种情况下，写集锁定和读集验证是本地的。

因此，不需要准备prepare消息和验证消息，并且主副本仅需要在解锁修改后的对象之前发送提交消息（给从副本），其中包含对从副本的缓冲写操作。此外，我们使用两种锁定模式：**首先将对象锁定为允许无锁读取的模式，而在更新对象之前（将提交消息传递到从副本之后），主对象将处于独占模式（也就是说，在事务启动到提交期间是可以无锁读取主副本对象的）。**单机事务通过减少消息数量并进一步减少由于锁定引起的延迟来提高性能。

### 3.5 无锁操作

FaRM提供了无锁读取，**该读取可与事务进行序列化**，并且使用单个RDMA读取执行，而无需使用远程CPU。即使应用程序与写入同一对象并发，也可以保证该应用程序观察到一致的对象状态。 FaRM依赖于缓存一致性DMA：它在对象标头的**第一个字**和每个缓存行（第一个除外）的开头中都存储对象的**版本号**。这些版本对应用程序不可见。

FaRM在读取和写入时自动转换对象布局。

lockFreeRead使用RDMA读取对象，并检查标头版本是否已解锁并匹配所有高速缓存行版本。如果检查成功，则读取将严格地与事务序列化。否则，在随机退避后重试RDMA。图8显示了跨越三个缓存行的对象的版本字段。

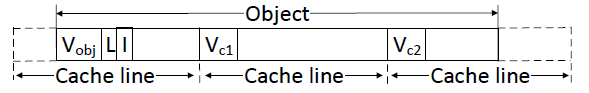


图8：无锁读取的版本控制。如果锁定字段L为零并且Vc1和Vc2与Vob j的低位匹配，则读取是一致的。化身I用于来检测对象是否在读取时被同时释放。

在事务提交期间，使用本地内存写入了对象。标头版本在准备阶段通过compare-and-swap锁定。我们使用标头版本中的**两个最低有效位**来对锁定模式进行编码。在提交阶段，通过首先**将特殊的锁定值写入高速缓存行版本，然后更新每个高速缓存行中的数据，最后更新高速缓存行版本和标头版本**，从而完成更新对象。这些步骤由内存屏障分开。在x86处理器上，编译器障碍足以确保所需的排序。由于DMA在x86上是高速缓存一致性的，因此任何RDMA读取都将按照由内存屏障强制执行的顺序观察每个高速缓存行内的内存写入。因此，对对象跨越的所有高速缓存行进行版本匹配可确保严格的可序列化性，以实现无锁读取。（需要看代码来确定逻辑）

我们使用64位的标头版本来防止回绕（就是位数足够多，防止溢出），但是高速缓存行版本仅保留该版本的最低有效l位以节省空间。我们之所以这样做，是因为执行写操作的时间有一个下限，并且我们中止需要花费比**上限更长的时间**的RDMA读操作，以确保读操作永远不会与产生**具有相同最低有效l位版本的两个连续写操作**重叠（适当设置时间可以减少冲突）。这依赖于时钟漂移的弱界限，而我们已经需要使用ZooKeeper来维持租约（关系）。本文的结果是在l = 16时获得的，但我们的测量结果表明，使用l = 8的配置对于复制已经足够。

**为了提供一致性，FaRM必须确保无锁读取不会访问由并发事务释放（free）的对象。** FaRM使用类型稳定性[19]来确保**对象元数据**保持有效，并使用化身检查[46]来检测何时释放对象。对象头具有64位化身（incarnation），该化身最初为零，并在释放对象时增加。 FaRM提供了128位胖指针，其中包括对象地址，大小和预期化身（incarnation）。应用程序检查无锁读取**返回的对象缓冲区中的化身（版本的意思吧）**是否与指针中的化身相匹配，以确保未释放对象。

FaRM可以重新使用释放的内存来分配相同大小的另一个对象，因为对象头中的化身仍然有效。为不同的对象大小重新使用内存需要更多的工作，因为对象标头可能会被任意数据覆盖。 FaRM实现了基于epoch的分配器的分布式版本[17]。它将end of epoch请求发送到所有计算机上的线程（我们将消息聚合到同一计算机/从同一计算机发送）。

**当线程收到此请求时，它将清除所有缓存的指针，开始一个新的纪元，并继续在新纪元中处理操作。**一旦在先前时期开始的所有事务和只读操作完成，线程便发送对请求的答复。

FaRM的API提供了将操作**括起来**的原语，以便能够检测正在进行的操作何时完成。

**在收到当前配置中所有计算机的响应后，可以重用内存**。该机制不会在很大程度上影响性能，因为它在后台运行并且仅在可用内存降至阈值以下时运行。

### 3.6 Hashtable

FaRM还提供了一个通用的键值存储接口，该接口被实现为共享地址空间顶部的哈希表。此接口的一个重要用途是作为根来获取给定键的共享对象的指针。

设计使用RDMA表现良好的哈希表类似于其他形式的**内存层次结构感知数据结构**设计：在实现良好的**空间效率**与执行常见操作所需的**最小RDMA数量和大小**之间取得平衡非常重要（即将RDMA看成I/O，希望的是读写用到最少的I/O）。理想情况下，**我们希望使用单个RDMA读取执行查找，这是最常见的操作**。我们认为跳房子（hopscotch）哈希[21]是实现该目标的一种有前途的方法，因为它保证了键值对位于可以用单个RDMA读取的较小的**连续内存区域**中（还是要确保空间的连续性）。这与基于cuckoo哈希的流行方法形成相反对比[40]，在该方法中，键值对位于几个不相交的区域中的一个。

**跳房子哈希表**中的每个存储桶都有一个邻域，其中包括该存储桶和随后的H -1个存储桶。跳房子散列可保持键值对存储在key存储桶（即，key 哈希到的存储桶）的邻居的不变性。要插入键值对，该算法会通过向前进行线性探测来寻找靠近键值存储区的空存储区。如果空存储桶位于键存储桶的附近，则键值对将存储在该存储区中。否则，该算法将通过在保留不变性的同时重复移动键值对来尝试将空存储桶向邻域移动（即腾出空间）。如果算法找不到空存储桶或无法保留不变式，则将调整哈希表的大小。

原始算法在使用H = 32 [21]的高占用率情况下的性能优于链式和布谷鸟cuckoo哈希表（其中占用率是插入的键-值对的数量与表中的插槽数量之间的比率）。不幸的是，使用大邻域时，RDMA的性能较差，因为它们会导致大量读取。例如，使用H = 32和64字节的键值对需要至少2 KB的RDMA读取，这比较小的RDMA读取要差得多（图2）。简单地使用小领域并不能很好地工作，因为这需要频繁调整大小并导致空间效率低下。例如，当H = 8时，原始算法的平均占用率仅为37％。

我们设计了一种新的算法，**链式关联的跳房子哈希算法**，通过将跳房子的哈希算法与链接和关联性相结合，实现了空间效率与用于执行查找的RDMA的大小和数量之间的良好平衡。例如，平均来说，在90％的占用率下，每次查找仅需要1.04个RDMA读取，H = 8。这比基于cuckoo哈希的技术要好[40]，后者需要以75％的占用率进行3.2 RDMA读取（如果在表中内联键值对，则为1.6）[37]。

新算法在每个存储桶中使用一条**溢出链**。如果插入内容无法将空存储桶移至正确的邻域，则会将键值对添加到key存储桶的溢出链中，而不是调整表的大小。这也使我们可以限制插入过程中线性探测的长度。该算法使用关联性来分摊跨几个键值对之间的链和FaRM对象元数据的空间开销。每个**bucket桶**都是一个带有H / 2个插槽的FaRM对象，用于存储键值对。该算法确保将键值对存储在键的存储桶或下一个存储桶中。溢出块还存储几个键值对（当前为两个）以提高性能和空间效率。

我们使用FaRM的API实现了新算法。哈希表在群集中的所有计算机上分片。每台计算机都分配分片shards（即FaRM的bucket阵列），并与其他计算机交换指向该分片的指针。我们使用一致性哈希来在各个分片之间划分hash值，以实现弹性。

**使用无锁只读操作执行查找。查找密钥k的方法是发出一个RDMA来读取k的存储桶b和下一个存储桶b + 1。如果在b或b + 1中找到k，则查找完成。否则，它将使用无锁读取来搜索b的溢出块链中的k。该链使用胖指针链接块，并查找胖指针中的化身数字与下一个块是否匹配。**如果没有，则重新开始查找。将大型键值对内联存储在存储桶中效率不高，因为这会导致大型RDMA。 **FaRM将大或可变大小的键值对存储为单独的对象，并在存储桶中存储键（或大键的哈希）和指向该对象的粗指针**。如果胖指针和对象中的化身编号不匹配，则重新开始查找。

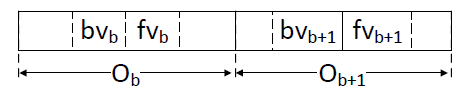


图9：连接版本，用于无锁读取相邻buckets.。如果两个对象分别一致且f vb = bvb + 1，则它们彼此一致。

3.5节中的版本检查可确**保每个bucket在读取时都保持一致**。但是，对于哈希表查找，**我们还必须确保附近的两个存储桶彼此一致**（这里只使用两个邻居，若是多个，则要保证多个桶的版本一致）。为此，我们为相邻的两个存储桶对添加**连接版本**，这意味着每个对象存储桶都存储一个**向前和向后**的连接**joint**版本（图9）。

如果相应的连接版本没有相同的值，则重新开始读取。更新相邻存储桶的事务将递增相应的连接版本。我们使用减少无锁读取的缓存行版本大小的相同技术，减少了连接版本的空间开销。本文的结果使用16位连接版本。

**我们通过将事务传送到存储相关分片的机器来优化插入、更新和删除。使用事务简化了实现，但查找要复杂得多。我们使用FaRM的API确保分片与溢出块并置，以便我们可以使用更高效的单机事务。**

我们如上所述实现插入。我们使用一种受平面组合(flat combining)[20]启发的技术，将对同一密钥的并发插入和更新组合到单个事务中。通过减少热键的并发控制和复制的开销，在偏斜的YCSB工作负载的实验中，这将吞吐量提高了4倍以上。**删除操作则尝试消除溢出链以减少用于查找的RDMA的数量**。他们总是将链中的最后一个键-值对移动到新释放的插槽，并在最后一个溢出块为空时释放它。否则，它们将增加其**化身编号**以确保查找观察到一致的视图。

FaRM的哈希表可确保线性化，并且性能良好。图10显示了当随机选择uniform密钥时，使用8字节key（占90％的占用率）和不同值大小的20机集群（请参见第4部分）上的进行查找的吞吐量。它显示了当值以不同的邻域大小内联和值存储在存储桶（hash table）外部（使用H = 8）时的结果。图11显示了同一实验中的空间利用率：这是键值对中的字节总数与哈希表使用的内存总量之间的比率。结果表明，对于小于等于128个字节的对象，H = 8或H = 6的内联值在吞吐量和空间利用率之间提供了良好的平衡。可以容忍低空间利用率以实现更高吞吐量的应用程序可以内联最大为320字节且H = 2的对象。**大于320字节的对象应存储在表的外部**。

