Carbink: Fault-Tolerant Far Memory

Carbink is a Pokémon that has a high defense score

总结

远内存系统允许应用程序透明地访问本地内存以及属于远程机器的内存。 容错是任何远存储器实用方法的关键属性，因为机器故障（计划内和计划外）在数据中心中很常见。 然而，设计一个对计算和存储都有效的容错方案是困难的。本文提出了具有容错能力的远存储系统Carbink，它使用纠删码、远端内存合并、单边 RMA 和可卸载奇偶校验计算来实现快速、存储高效的容错。 与最先进的远程内存容错系统 Hydra 相比，Carbink 的尾延迟降低了 29%，应用程序性能提高了 48%，内存使用率最高提高了 35%。

背景

内存密集应用对内存的需求越来越高，但是物理内存的扩展受到限制。Google和Alibaba的测试报告指出，数据中心服务器平均的内存使用率为60%，因此，数据中心中有大量的闲置内存。因此，目前采用高速网络，访问远端内存。

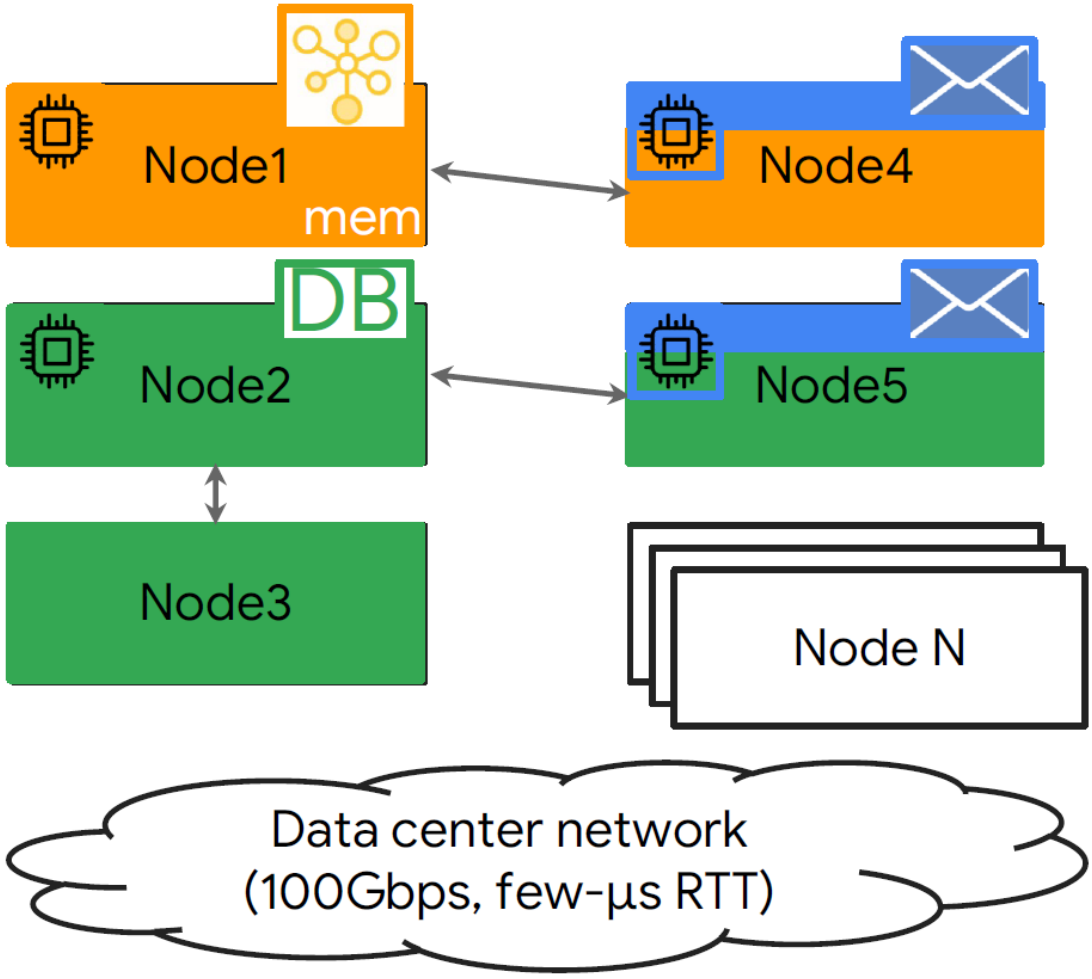


图 1. 远端内存架构

目前访问和管理远端内存主要有下列的方式：（1）修改操作系统抽象层，让操作系统将far memory当成swap设备，在应用层代码上，还是采用标准的c++指针用以访问far memory中的对象；（2）第二种方法则是修改应用代码，提供智能指针用来访问远端内存中的对象，并在本地节点维护远端内存映射关系。第一种方法虽然不需要对应用代码作太大修改，但是修改OS内核代码开发难度较高，且需要适应OS kernel的演化，因此，本文采用方法2，实现针对远端内存对象访问的智能指针，并实现一个全局的内存管理运行时程序memory manager。

问题&动机

关于远端内存，目前研究工作较多，但是很少有工作考虑远端内存系统的容错能力。而在数据中心中，系统故障经常发生，因此本文主要设计了一个具有高容错能力的远端内存系统。

1.内存错误模型

计划内错误：数据中心管理员进行的计划内调度，如软件升级，硬盘挂载，格式化，系统升级/重装等。

非计划内错误：内核错误，硬件故障，电源中断等。

本文没有考虑网络传输错误，仅局限于内存节点和计算节点的错误。

2.SOTA工作缺陷

为了解决内存错误问题，目前主流的方式为副本技术和纠删码技术。常见的三副本技术保留了数据的三个副本，因此导致3X容量开销。纠删码技术则是一种低容量开销的容错技术，如RS（4,2）码容量开销为1.5X，单核可实现4GB/s的编码带宽。因此，从容量开销角度考虑，目前主要采用纠删码方案，当然，针对某些应用场景，副本+纠删码一起结合使用。

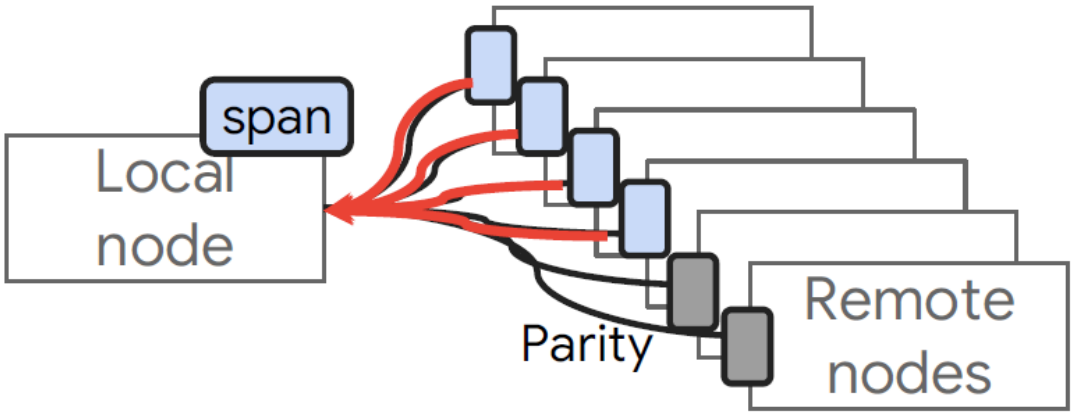


图 2. Hydra纠删码编码方式

Hydra的方式是针对一个page对齐的span，一个span中包含多个object，并且swap in/out的粒度为span。Hydra对一个span进行编码，编码技术需要进行数据分块，并将多个数据块存放在多个memory node，校验数据块存在校验memory node中。（Q：为什么需要存在多个memory node中？A：存在一个node，这个node崩了，没法进行数据恢复）显而易见，采用这种方式，**好处**是编码粒度与swap in/out粒度相同，当一个span无效时，对应的编码的span数据及其校验数据无效，无span碎片。**坏处**就是进行数据重构需要多次请求，一是会带来带宽压力，二是会产生尾延迟，部分请求时间较长，且当所有对应的node数据都读到计算节点时，计算节点才能进行恢复。

设计

memory manager用来进行远端内存池的管理，主要有allocate和register操作，当远端内存注册之后，compute node即可进行分配还未分配的内存。同理还有deallocate和deregister操作。

Carbink对内存节点和计算节点并没有特殊的硬件需求，当数据中心中的一个机器运行了carbink memory host daemon，其即可为memory node，当任一机器使用了carbink runtime，即可成为compute node。

Carbink在compute node中存放remotable pointers用来访问远端的object，其中swap in的粒度为span（size为N*page_size），swap out粒度为spanset（多个span，本文设置为4）。

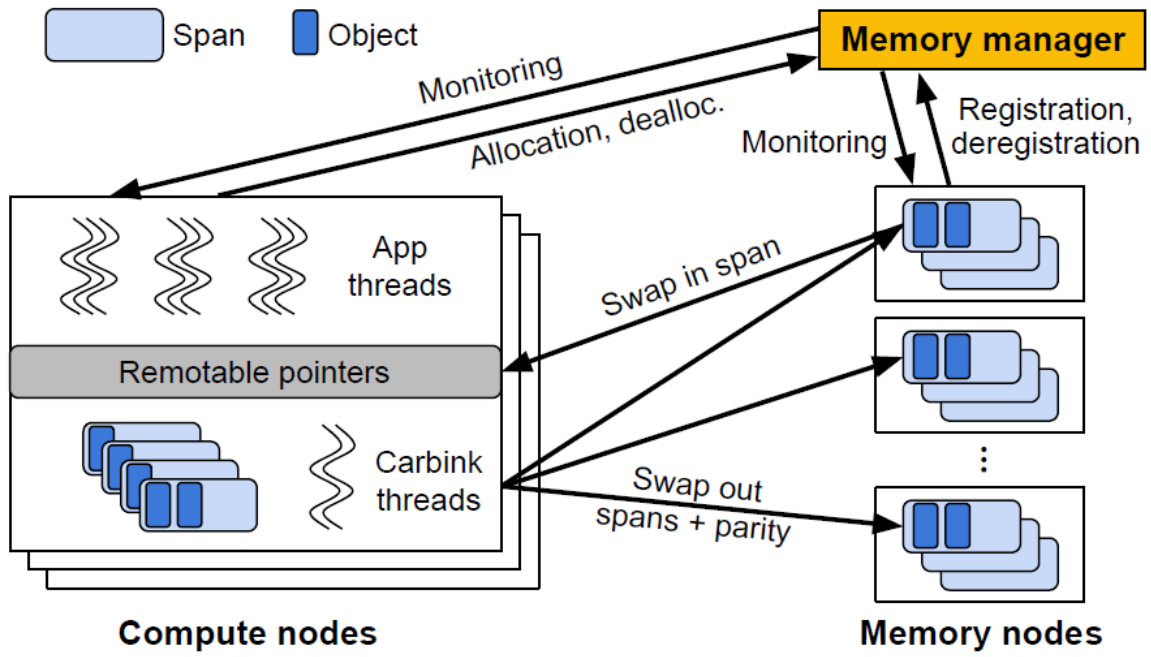
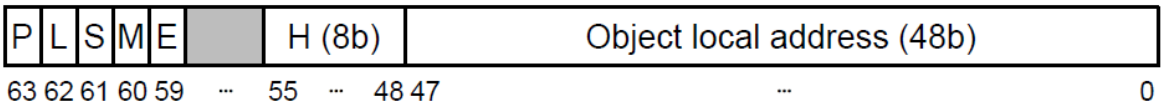


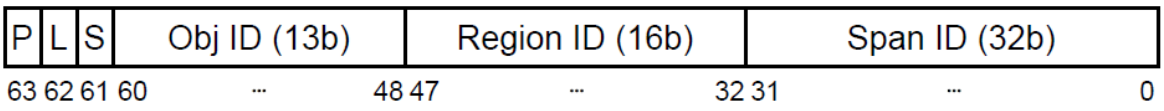
图 3. Carbink总体架构

1.Remotable Pointers

对c++中的只能指针进行重新编码。Carbin对c++中的unique_ptr和shared_ptr进行扩展，得到RemUnique_ptr和RemShared_ptr。当一个对象被移动或者从local 中驱逐，为了对指针进行更新，每个object有一个reverse pointer，指向对应的指针并对其进行更新。RemUnique_Ptr表示对一个对象的引用，RemShared_Ptr表示一个对象被多个指针引用，为了更新多个RemShared_Ptr，采用链表方式维护RemShared_Ptr。



(a) Local object.



(b) Far object.

Field	Meaning
Present	Is the object in local RAM or far RAM?
Lock	Is the object (spin)locked by a thread?
Shared	Is the pointer a unique pointer or a shared pointer?
Moving	Is the object being moved by a background thread?
Evicting	Is the object being evicted by a background thread?
Hotness	Is the object frequently accessed?

(c) Field semantics.

图 4. 对象智能指针设计

2.Span-Based Memory Management

图5展示了对不同大小object进行编码的方式，第一种采用填充，会造成大量内存浪费，第二种进行分块，对小object不友好，会有大量的元数据开销。

Carbink采用span粒度对数据进行管理。span是page-aligned，其将大小相似的object放到一个span中。local memory分配采用span粒度，span并不是一个固定大小，而是8KB-aligned，如2MB，4MB等。far memory分配采用region粒度，如1GB。内存分配的元数据存储在compute node中。

此外，local memory中，对cold objects和dead objects进行组织，将这些objects重组织为span，并且多个span组成一个spanset，按照spanset粒度swap out。

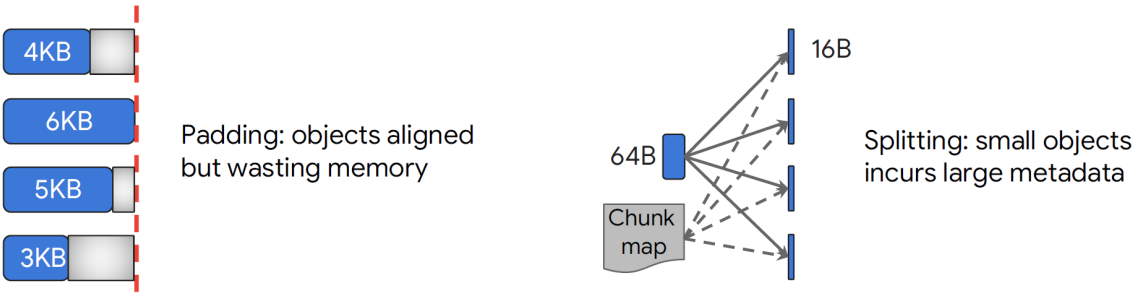


图 5. 对不规则大小的object进行编码

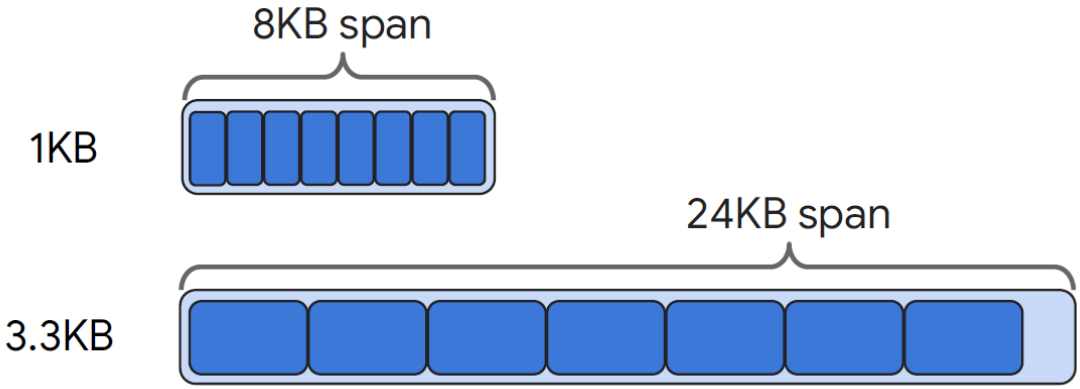
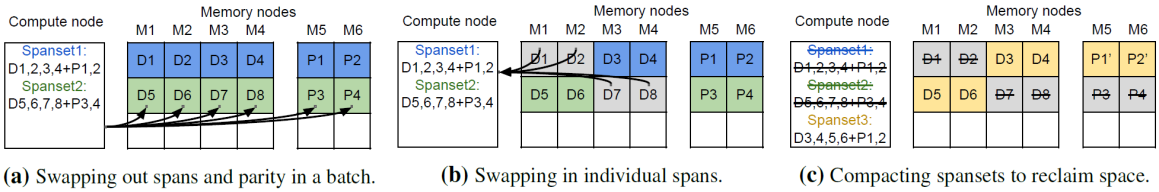


图 6. Carbink span粒度组织

3.Fault Tolerance via Erasure Coding

该工作的主要目标就是减少EC系统的尾延迟，优化性能，其选择spanset粒度进行编码，从而以span粒度swap in时，尾延迟低，但是swap in和swap out粒度不匹配，造成较大的far memory消耗，因此，本文后续主要是目的是如何回收far memory中的内存。



(a) Swapping out spans and parity in a batch. **(b)** Swapping in individual spans. **(c)** Compacting spansets to reclaim space.

图 7. 对两个可以互补的spanset进行合并，并提供了两种模式计算parity，local和remote

4.Failure Recovery

对于计划failures，memory manager停止对相应memory node的内存的register和allocate。

对于计划外failures，则采用相应的纠删码机制进行恢复，当且仅当错误在纠删码的错误恢复能力之内才可，此外，当M1错误时，计算节点对D1进行恢复，若此时计算节点想读取D1中的object，此时采用降级读机制，计算节点读取D2，D3，D4，P1，恢复D1，之后读取D1。

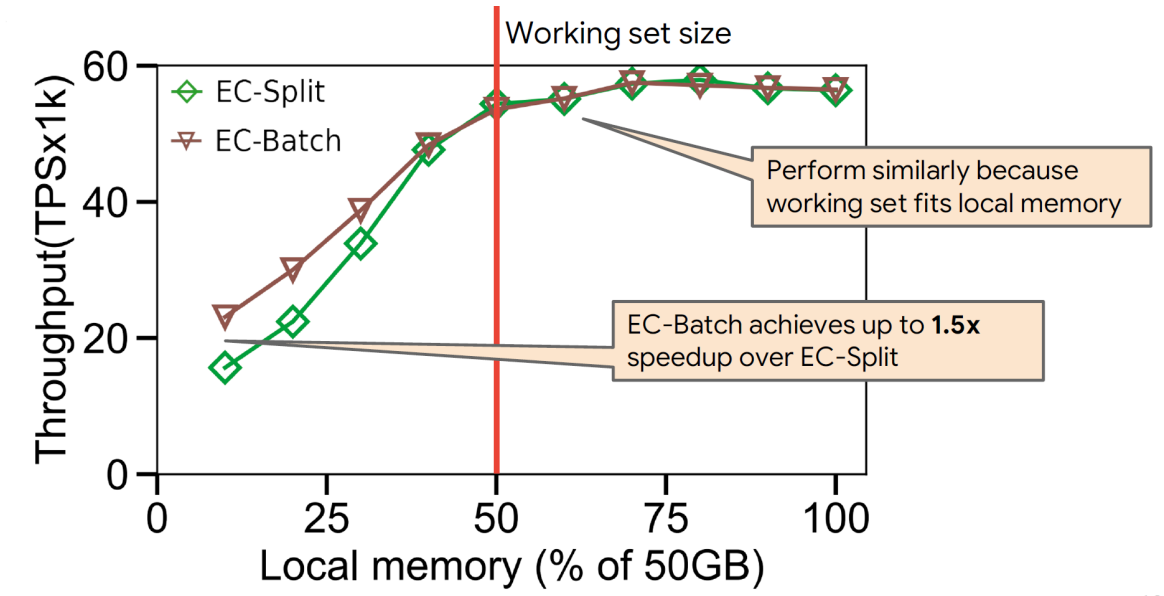
5.Thread Synchronization

一个对象可能被多个进行访问或者修改，如应用线程，filter线程（local memory中object移动），evict线程（驱逐到far memory）。针对多个线程间资源的竞争，通过RCU锁实现线程间同步，同时根据对象的spinlock来获取object访问。

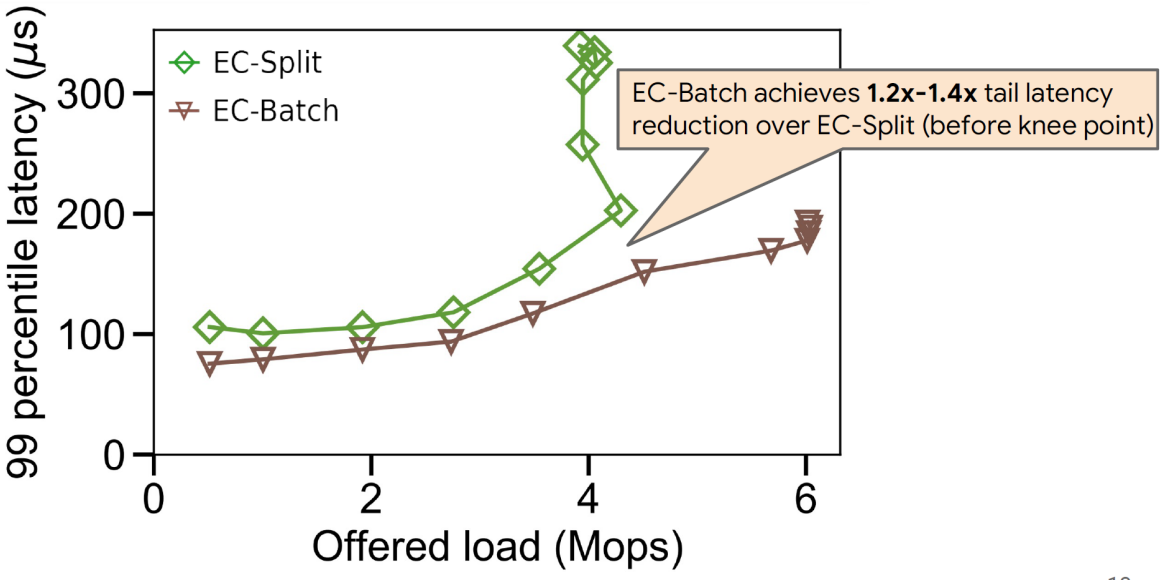
测试

测试采用了8台机器，放置于一个机架中，其中包含一个计算节点，7个内存节点，针对于RD(4，2)编码，4个数据节点，2个校验数据节点，一个memory node用以存储恢复的数据。

带宽测试（kv store）



尾延迟测试（microbenchmark）



评价&思考

优点

1.相较于SOTA工作，在尾延迟和性能方面有较大提升，且恢复时间接近副本技术。

2.方案简单有效

3.实验做了很多，比较扎实

缺点

1.问题也很明显，更大粒度的编码，导致了far memory中需要进行空间回收，且提出的compaction操作，虽然能够减少重计算新的spanset的校验数据的网络带宽压力，但是推迟的compaction操作会影响内存回收效率。