

**研究生学位论文开题报告**

**Graduate Thesis/Dissertation Proposal**

|  |  |
| --- | --- |
| **学号 Student ID** | 120037910002 |
| **姓名 Name** | 贾兴国 |
| **学生类别 Degree Program** | 学术型硕士生 Academic Master Student |
| **学习形式 Study Mode** | 全日制 Full-time |
| **导师 Supervisor(s)** | 戚正伟 |
| **论文题目 Thesis title** | 面向虚拟化环境的内存池系统设计与优化 |
| **学院 School** | (037)电子信息与电气工程学院(软件学院) |
| **专业 Major** | 计算机科学与技术(081200) |
| **开题日期 Date** | 2021-12-01 |
| **开题地点 Venue** | 上海交通大学（闵行校区） |

填 报 说 明

**Instruction**

1. 校本部研究生的开题报告应通过[数字交大](http://my.sjtu.edu.cn/)在线提交申请，填写本表并上传系统。特殊情况下经研究生院事先同意，可不上传系统，并使用《上海交通大学研究生论文开题评审表》完成评审。

The application for thesis/dissertation proposal should be submitted online through [My SJTU](http://my.sjtu.edu.cn/). The student shall filled this form and upload it in the system. Under special circumstance, this form does not need to be uploaded and the review can be proceeded with the review form with prior consent from the graduate school.

1. 开题报告为A4大小，于左侧装订成册。各栏空格不够时，请自行加页。考核前提前一周送交导师、评审专家审阅。

This form should be printed with A4 papers and bound together on the left. If the space left is not enough, please feel free to add extra pages. The print version shall be sent to the supervisor, and the review committee members for review at least one week before the oral presentation.

1. 博士生导师可以根据博士生学位论文选题情况自行确定是否进行开题查新，博士学位论文开题查新报告应由查新工作站提供。

The supervisor should decide, based on the proposed topics, whether a novelty assessment report is needed or not, which should be conducted by an authorized novelty assessment department.

1. 开题报告通过后，定稿版开题报告由研究生、导师各存档一份，无需上传系统。

Upon passing the proposal, the final version of this report shall be archived by the graduate student and his/her supervisors for future reference.

1. 医学院研究生如果以函评形式开题，开题地点请填写“函评”，专家组组长签名由导师签名。

For students in the School of Medicine, if the dissertation proposal is conducted via peer review, the “Proposal venue” shall be filled with “peer review” and the “Signature of Committee Chair” shall be signed by the supervisor.

|  |  |
| --- | --- |
| 论文题目  Proposed Title | **面向虚拟化环境的内存池系统设计与优化** |
| 研究课题来源  Source of Research Project | 请在合适选项前画√ Please select proper options by “√”.  国家自然科学基金课题 NSFC Research Grants  国家社会科学基金 National Social Science Fund of China  国家重大科研专项 National Key Research Projects  其它纵向科研课题 Other Governmental Research Grants  企业横向课题 R&D Projects from Industry  自拟课题 Self-proposed Project  其它 Other |

1. **请综述课题****国内外研究进展、现状、挑战与意义，可分节描述。博士生不少于10,000汉字，硕士生不少于5,000汉字。请在文中标注参考文献。 Please review the frontier, current status, challenges and significance of the research topic. The citations should be marked in the context and listed in order at the end of this section. No less than 8,000 words for doctoral students and 4,000 words for master students if written in English.**

近年来，摩尔定律已经走向终结，芯片性能的增速已经放缓，这意味着单位正本所能购买到的硬件性能不再继续增长。然而，随着大数据计算、人工智能、图计算等新应用的出现，越来越多的数据将会在内存中被处理；同时，用户对低延迟服务的需求越来越高，于是服务端更加需要将数据放在内存中以保证快速响应[1]。各个方面的需求导致内存容量需求的爆炸性增长。然而，在数据中心中，内存的成本已经成为总成本中较大的一部分，例如在微软的Azure云数据中心中，内存的成本占到了总拥有成本（TCO，Total Cost of Ownership）的50%[2]，且未来每GiB的内存价格依然有增长的可能。除了应用如SCM（Storage Class Memory，存储级内存）等硬件手段扩充内存空间外，提高内存空间的利用率、优化内存管理方式是降低数据中心成本的重要途径。在现如今的数据中心中，内存空间利用率尚有待提高。如Google提供的数据表明，Borg集群的内存使用率从2011年的约40%提升到了2019年的约60%[3]；阿里的数据表明，集群内存使用率仅有不到60%[5]。即使内存利用率如此低下，集群中的任务仍然会由于资源不足而被停止执行[4]。若能够利用空闲内存，则数据中心的服务提供者将获得巨大的收益。

具体而言，云数据中心向用户提供虚拟机（VM，Virtual Machine）， 每个虚拟机对不同资源有不同的需求（如CPU核心数，内存容量大小，磁盘容量大小等），其需求形成一个需求向量。于是，为了将多个虚拟机调度运行在同一个物理机上，并充分利用物理机的所有资源，我们需要解决一个多维装箱问题[2, 4]。例如，将CPU需求高于内存需求的多个VM运行在同一个物理机上，则物理机的CPU资源将很快达到饱和，而内存资源则会有部分空闲；为了继续利用这部分空闲内存资源，我们不得不将更多的虚拟机运行在这个物理机器上，使得CPU资源过饱和，vCPU（虚拟CPU）之间相互频繁地抢占，造成虚拟机服务质量的严重下降。我们将这类剩余内存无法被虚拟机利用的现象称为内存滞留（Memory Stranding）。在Azure上的研究表明，高达25%的内存会受到内存滞留现象的影响而无法使用[2]。

内存滞留问题背后的根本原因是单一服务器（Monolithic Server）的计算模型，即不同种类硬件资源紧耦合的计算模型。单一服务器中的CPU、内存等资源的配比是固定的，无法向虚拟机灵活分配不同比例的物理资源，且如果其中一种资源出现故障（如内存控制器故障），则整个服务器的其他计算资源将不可使用（如CPU、存储等资源）。为了解决单一服务器模型中存在的问题，美国普渡大学的研究者提出了硬件资源分离的架构，并在其上实现了分离内核架构的LegoOS操作系统[4]。在这种硬件资源分离架构中，每个主机只对外提供单一的一种物理资源，而不是和在单一服务器架构中一样向外提供全部种类的硬件资源，每个主机称为一个组件（Component），主机之间用高速网络连接起来，为同一个操作系统提供运行所需的全部硬件资源。每个Component都有自己的控制器（Controller）运行着资源管理器（Monitor）软件，并有网络接口与外界通信。这样的分离式硬件架构将不同种类的硬件相互解耦，方便上层软件以任何比例使用物理资源，从而消除了资源滞留的现象，提高了资源利用率，同时提高了系统的容错能力，由于单个Component的崩溃不会使得整个系统停止工作。除此之外，新硬件可以方便的以一个网络节点加入整个系统，而不是加入集群中的某一个节点，从而避免了现有物理节点不兼容新硬件而无法快速将新硬件整合入现有系统的问题。目前有三种Component，包括CPU资源pComponent，内存资源mComponent，以及存储资源sComponent。运行在Component之上的分布式操作系统称为Splitkernel（分离式内核）架构的操作系统，LegoOS是其一种实现方式。LegoOS为用户提供了vNode的抽象，vNode类似一个虚拟机，它具有独立的ID，独立的IP地址，以及独立的存储器挂载点。用户可以在一个vNode中运行多个进程；每个vNode运行在多个Component之上，而每个Component也可以为多个vNode提供物理资源。这意味着，在pComponent上运行的进程为了访问其内存数据，需要经过网络访问远程的mComponent。这样的架构势必会因为巨大的网络开销导致性能严重下降，但在现有硬件以及软件配合的基础上，仍然保证了性能。对于内存资源mComponent的远程访问，LegoOS均使用高带宽低延时的RDMA网络进行访问，同时在CPU资源端维护了一个ExCache（Extended Cache，扩展缓存），在一个较小的主存空间中（4GiB）保存了mComponent的数据缓存，保证热数据的快速访问。为了提升内存利用率以及内存子系统的并发能力，每个进程的地址空间可以跨越多个mComponent。总而言之，分离式硬件架构彻底解决了资源滞留的问题，同时获得了易扩容、高容错等巨大的优势，相比于单体（Monolithic）服务器架构是一种巨大的创新，LegoOS也因此获得了2018年操作系统顶级学术会议OSDI的最佳论文奖。

具体而言，保证LegoOS成功的因素有硬件和软件两个方面。在硬件方面，网络性能的发展日新月异。远程内存直接访问技术（RDMA，Remote Direct Memory Access）如Infiniband相比于传统的以太网加TCP/IP协议有了更低的延时以及更高的吞吐量，根据最近的一份报告表明，Infiniband的带宽将要达到200Gbps（Gigabyte per second，吉比特每秒），延迟将降低至600纳秒以下[12]，而内存总线带宽大约在400Gbps到800Gbps左右，这说明远程内存访问的带宽仅为本地内存带宽的四分之一到二分之一。这说明，将所有内存放在远程进行访问所造成的开销正在快速降低，而内存则遇到了内存墙问题[7]，其带宽的发展已经放缓，网络的速度有望超过本地内存的速度。除了远程内存直接访问技术外，还有诸如CXL[8]、CAPI[9]、CCIX[10]、Gen-Z[11]等技术，使得远程资源的访问更加快速且方便，这些硬件保证了LegoOS所提出资源分离模型的成功。在软件方面，LegoOS没有复用现有开源操作系统的代码，而是重新构建了一个操作系统，针对跨网络访问进行了多重优化，将网络栈压缩到最短，从而减小了分离式架构中需要跨网络访问硬件资源的劣势。在远程内存方面，LegoOS在CPU端维护了一个虚拟寻址的基于主存的ExCache，在内存端实现了一种高效的双层虚拟内存管理机制，保证了前端的快速内存访问和内存的均衡分配，从而消除了内存访问热点，同时优化了内存空间的使用。由于内存分配需要经过网络，LegoOS在内存分配上尽可能采取“按需”的策略，如分配全部置0的匿名页面时，只要应用不修改该页面的内容，那么LegoOS就只在ExCache中分配该页面。综上所述，只有软件和硬件相结合，软件充分考虑硬件的特征，并利用硬件的优势，取长补短，才能达到最大的优化效果。

除了内存与处理器资源分离外，LegoOS还支持存储资源的分离，即支持远程访问sComponent的磁盘上的数据。为了保证sComponent的功能与其他部分严格解耦，所有的文件访问请求都必须包含完成该请求的全部信息，如完整的文件路径、访问的绝对偏移。同时，为了保存sComponent的磁盘缓存数据，mComponent还需要配合sComponent，若pComponent需要访问sComponent上的一个文件，则会根据pComponent端缓存的文件信息找到缓存数据所在的mComponent，若找到，则可以快速地返回数据，若没有找到，则mComponent将请求转发给sComponent，并从其中得到磁盘上的数据，存储在mComponent中。由于目前的关注点在提升内存资源的使用率上，故我们暂时不借鉴LegoOS分布式存储的实现。该实现只是作为验证存储资源也可以与计算资源分离而做的一个实验。

由于内存资源利用率问题的重要性，学术界对远程内存的研究热情一直十分高涨，有很多不同于LegoOS、仅构建远程内存的系统。其中基于软件的远程内存系统大致可以根据是否修改应用分为两类。

**基于OS、Hypervisor的透明远程内存系统**

不修改应用的方案主要是在OS（Operating System，操作系统）或虚拟机管理器（Hypervisor）中实现远程内存的逻辑，使得运行在用户态的应用感知不到底层内存架构，从而无需修改应用，即可获得远程内存的优势。这样的方案在更低的抽象层实现了远程内存的管理逻辑，对现有单机应用有着广泛的兼容能力，且可以复用OS或者Hypervisor已经实现好的功能，工程量较小，所以学术界中大量的远程内存系统均基于这个方案。首先是2017年发表在NSDI上的Infiniswap工作[12]，其动机是现有远程内存系统均需要修改硬件架构或修改应用的实现，使得远程内存系统不能广泛应用。于是，Infiniswap基于现有OS的内存交换系统（Swap），当本地内存资源不足时，为应用提供了新的换出本地内存的途径，即远程内存，并且为了保证容错性，在同步地换出到远程内存时异步地换出到本地磁盘。其中，远程内存的访问使用了RDMA网络，每个机器上加载一个块设备，块设备中的每个部分映射到不同的远程机器，以4KiB内存页的粒度进行换出，复用了Linux内核的block I/O系统。远程节点上运行一个用户态进程Infiniswap Daemon，负责管理远程节点上的本地内存，并响应块设备发出的内存换出请求。Infiniswap可以充分利用远程内存，使得集群内存利用率提升2.36倍。相似的系统还有Remote Regions[13]，只是该系统对应用提供的接口、内部实现方式与Infiniswap不同。它向应用提供了VFS（Virtual File System）的接口，在文件系统中实现访问远程内存的逻辑，从而掩盖下层不同的远程内存硬件的语义（如Infiniband verbs），方便用户程序使用远程内存。基于块设备和文件系统的远程内存系统有较为复杂的调用栈，虽然通过复用其已经实现的功能，但会执行一些与远程内存无关的代码逻辑，并非为远程内存定制，这部分有较大的优化空间。于是，后序工作有对基于Swap的远程内存的进一步优化。Fastswap[14]观察到在请求远程页面时，CPU会等待较长时间，浪费了CPU时间片，于是提出了预取的方式以掩盖较长的网络I/O时间，在发起一次对远程页面的网络请求后，将根据一些预测的结果向远端预取接下来的几个内存页，当所有网络请求发送完毕后，在等待第一个页面的请求结果。这种预取的模式可以充分利用网络带宽，减小远程内存引起的性能下降，测试表明相比于Infiniswap可以获得2.54倍的内存带宽提升，同时内存带宽的利用也有显著提高，对应用性能有较大的益处。Leap[15]注意到磁盘Swap的代码路径对于远程内存系统是过长且不必要的，并且掩盖了底层低延迟高带宽网络的优势。Leap设计了一套不同于磁盘换出的数据通路，并且使用了一种跳跃式的内存页面预取算法，对应用将要访问的远程内存页面进行了精确的预取。相比于Infiniswap，得益于缩短的数据通路以及智能的预取算法，内存密集型应用访问远程内存页的尾延迟的中位数下降了104.04倍，应用性能也提升了10.16倍。

除远程内存的探索方向之外，还有两个工作探索了其他可以挖掘的内存空间，形成了另一种软件定义（Software-defined）的远程内存，也基于OS的修改，对应用透明。其中一个工作是发表在ASPLOS’19上的zswap系统，该系统认为物理的远程内存节点存在可靠性低的问题，例如可能受到网络断开的影响，系统无法正常工作，且分布式系统部署较单机系统更加困难[16]。于是，zswap用压缩的内存代替了远程内存硬件，将内存压缩作为Swap的一个数据分支，将不常使用的冷数据页用压缩算法进行压缩，减小其空间占用，并保存在本地节点。为了精准探测应用的冷页，zswap还加入了机器学习算法，不断地根据应用的访存情况动态修改冷页探测算法的参数，从而实现在数据中心中大规模的精确冷页探测。zswap自2016年已经部署在Google的WSC（Warehouse-Scale Computer，仓储规模计算机），降低了67%的内存容量需求，但只压缩了20%的内存页，且该部分被压缩的内存页只有6微秒的访问延时，相比于Infiniswap硬件远程内存系统毫秒级别的延迟大幅下降，从而在不影响应用性能的前提下大大降低了WSC的总拥有成本。另一个工作是发表在IEEE TC’20上的XMemPod系统[18]。该论文的作者观察到，由于虚拟机和宿主机之间的语义鸿沟，宿主机分配给虚拟机的部分内存不一定被使用。在访问开销较大的远程内存前，XMempod会在本地内存中寻找虚拟机尚未使用的物理内存，这部分“远程”内存的访问速度和普通的本地内存相同。XMemPod挖掘了内存虚拟化中语义鸿沟引起的“内存滞留”问题，相比于SOTA（State-of-the-art，现有最高水平）的远程内存系统，运行Spark，Apache Hadoop，Memcached等内存密集型应用的吞吐量增加了1.7到14倍，同时降低了24%的远程内存带宽消耗。

**基于运行时的远程内存方案**

不修改应用的方案虽然对应用透明，容易将现有应用运行在远程内存系统上，但依旧不能达到极致的性能。其中的原因有4KiB页粒度的内存管理，造成了写放大问题，以及复杂的操作系统调用栈造成的开销。对于应用透明这一优势是一把双刃剑，虽然不需要上层应用修改，但难以捕获上层应用语义，无法与应用逻辑相互协作提高性能。修改应用的方案在更高的抽象层进行修改，为应用编写者提供新的专用API，从而调用下层运行时，完成特定的功能。由于这个探索思路较为新颖，且难以集成到现有环境中，现有的工作数量较少。最具代表性的是AIFM运行时[17]，提出了remoteable pointer（可移动指针），由运行时决定该指针指向的内存数据在本地内存还是远程内存。在可移动指针的基础之上，C++开发者可以设计出使用远程内存的数据结构，扩展了数据结构可以使用的内存。可移动指针底层的运行时负责数据对象的换入换出、预取，以及内存对象的驱逐。相比于基于OS的远程内存系统，AIFM换入换出、预取的对象是一个数据结构而非内存页，精准操作应用真正需要的内存，消除了写放大问题。另一项工作是针对JVM（Java Virtual Machine，Java虚拟机）的修改，实现了分布式JVM，称为Semeru。这项工作观察到，管理型语言除应用逻辑外，还会进行垃圾回收操作，这一项操作通常不具有空间局部性，之前基于缓存、预取优化的远程内存系统将无法高效运行基于管理型语言的应用。于是，Semeru通过将JVM扩展到多个内存节点，无需修改JVM之上的应用，应用逻辑在CPU节点上运行，使用多个内存节点保存Java应用的对象。其中垃圾回收操作被卸载到内存节点上执行，内存节点上的垃圾回收只访问本地内存中的对象，从而将计算与内存拉近，降低了网络开销。由于Semeru更加贴近Java应用的逻辑，Semeru相比于基于OS页的远程内存系统更优的性能。

参考文献 Reference：

1. J. Yang, Y. Yue, and K. V. Rashmi, “A large scale analysis of hundreds of in-memory cache clusters at twitter,” in 14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020. USENIX Association, 2020, pp. 191–208. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/yang
2. H. Li, D. S. Berger, S. Novakovic, L. Hsu, D. Ernst, P. Zardoshti, M. Shah, I. Agarwal, M. D. Hill, M. Fontoura, and R. Bianchini, “First-generation memory disaggregation for cloud platforms,” CoRR, vol. abs/2203.00241, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2203.00241
3. M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, “Borg: the next generation,” in EuroSys ’20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020, A. Bilas, K. Magoutis, E. P. Markatos, D. Kostic, and M. I. Seltzer, Eds. ACM, 2020, pp. 30:1–30:14. [Online]. Available: https://doi.org/10.1145/3342195.3387517
4. Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, “Legoos: A disseminated, distributed OS for hardware resource disaggregation,” in 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018, A. C. Arpaci-Dusseau and G. Voelker, Eds. USENIX Association, 2018, pp. 69–87. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/shan
5. Alibaba. Alibaba production cluster trace data. [Online]. Available: https://github.com/alibaba/clusterdata
6. Mellanox. Connectx-6 single/dual-port adapter sup- porting 200gb/s with vpi. [Online]. Avail- able: http://www.mellanox.com/page/products dyn?product family=265&mtag=connectx 6 vpi card
7. B. M. Rogers, A. Krishna, G. B. Bell, K. V. Vu, X. Jiang, and Y. Solihin, “Scaling the bandwidth wall: challenges in and avenues for CMP scaling,” in 36th International Symposium on Computer Architecture (ISCA 2009), June 20-24, 2009, Austin, TX, USA, S. W. Keckler and L. A. Barroso, Eds. ACM, 2009, pp. 371–382. [Online]. Available: https://doi.org/10.1145/1555754.1555801
8. N. Shenoy. A milestone in moving data. [Online]. Available: https://newsroom.intel.com/editorials/milestone- moving- data
9. O. consortium. Welcome to opencapi consortium. [Online]. Available: https://opencapi.org
10. CCIX. Download the evaluation version of the ccix® base specification now. [Online]. Available: https://www. ccixconsortium.com/
11. G.-Z. Consortium. About the gen-z consortium. [Online]. Available: https://genzconsortium.org
12. J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, “Efficient memory disaggregation with infiniswap,” in 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017, A. Akella and J. Howell, Eds. USENIX Association, 2017, pp. 649– 667. [Online]. Available: https://www.usenix.org/conference/ nsdi17/technical- sessions/presentation/gu
13. M. K. Aguilera, N. Amit, I. Calciu, X. Deguillard, J. Gandhi, S. Novakovic, A. Ramanathan, P. Subrahmanyam, L. Suresh, K. Tati, R. Venkatasubramanian, and M. Wei, “Remote regions: a simple abstraction for remote memory,” in 2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018, H. S. Gunawi and B. Reed, Eds. USENIX Association, 2018, pp. 775–787. [Online]. Available: https: //www.usenix.org/conference/atc18/presentation/aguilera
14. E. Amaro, C. Branner-Augmon, Z. Luo, A. Ousterhout, M. K. Aguilera, A. Panda, S. Ratnasamy, and S. Shenker, “Can far memory improve job throughput?” in EuroSys ’20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020, A. Bilas, K. Magoutis, E. P. Markatos, D. Kostic, and M. I. Seltzer, Eds. ACM, 2020, pp. 14:1–14:16. [Online]. Available: https://doi.org/10.1145/3342195.3387522
15. H. A. Maruf and M. Chowdhury, “Effectively prefetching remote memory with leap,” in 2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020, A. Gavrilovska and E. Zadok, Eds. USENIX Association, 2020, pp. 843–857. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/al-maruf
16. H. A. Lagar-Cavilla, J. Ahn, S. Souhlal, N. Agarwal, R. Burny, S. Butt, J. Chang, A. Chaugule, N. Deng, J. Shahid, G. Thelen, K. A. Yurtsever, Y. Zhao, and P. Ranganathan, “Software-defined far memory in warehouse-scale computers,” in Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019, I. Bahar, M. Herlihy, E. Witchel, and A. R. Lebeck, Eds. ACM, 2019, pp. 317–330. [Online]. Available: https://doi.org/10.1145/3297858.3304053
17. Z. Ruan, M. Schwarzkopf, M. K. Aguilera, and A. Belay, “AIFM: high-performance, application-integrated far memory,” in 14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020. USENIX Association, 2020, pp. 315–332. [Online]. Available: <https://www.usenix.org/conference/osdi20/presentation/ruan>
18. W. Cao and L. Liu, “Hierarchical orchestration of disaggregated memory,” IEEE Trans. Computers, vol. 69, no. 6, pp. 844–855, 2020. [Online]. Available: https://doi.org/10.1109/TC.2020.2968525
19. **课题研究目标、主要研究内容和拟解决的关键问题。 Research objectives, main contents and key issues to be solved.**

**研究目标**

实现一个面向虚拟化环境的内存池系统，能够将内存池中的内存分配给虚拟机，从而提升内存的利用率，同时保证虚拟机性能。

**主要研究内容**

1. 设计一个内存池系统，使得虚拟机和普通应用能够从中分配物理内存，提高物理内存的利用率；
2. 通过内存页亲和性放置、线程调度、通道拥塞避免等手段进一步优化内存池性能；

**拟解决的关键问题**

1. 打通虚拟机与物理机之间的语义鸿沟，使得物理机可以观察到虚拟机真实使用的物理内存，从而降低虚拟机空闲内存占用；
2. 降低远程内存访问延迟，同时降低因为远程内存访问造成的网络开销；
3. **拟采取的研究方法、****研究方案及其可行性分析。Research methods and research scheme to be adopted and feasibility analysis.**
4. 打通虚拟机与物理机之间的语义鸿沟：通过半虚拟化的方法，更改客户机物理内存分配逻辑，使之与宿主机相互配合完成内存分配。具体方式为：在宿主机OS启动时，维护一个内存池的抽象，在客户机OS启动后，将其内存分配系统同时接入该内存池，后分配物理内存时即从该内存池中进行分配；
5. 降低远程内存访问延迟，同时降低因为远程内存访问造成的网络开销：通过分析线程与内存页的关系，首先将访问相同部分内存的线程调度到相同的节点，从而降低内存同步带来的开销；再通过内存页面移动的方式，平衡不同内存节点之间的访存负载，达到最高的访存效率。可参考相关Linux内核对非一致性内存访问架构优化的补丁，从中借鉴，或设计新的内存管理方式，优化内存与线程的放置；
6. 主要研究步骤：首先实现一个宿主机客户机通用的内存池系统，再通过测试探究其性能瓶颈，对瓶颈进行有针对性的优化设计
7. **课题的创新点 Novelties of the proposed topic.**
8. 打通虚拟机与物理机之间的语义鸿沟：现有系统尚未有虚拟机、物理机通用的物理内存分配方式，导致虚拟机的地址翻译需要经过双层页表，同时，也无法得知虚拟机内部的内存使用情况，导致虚拟化环境下内存的浪费；
9. 降低因为远程内存访问造成的网络开销：尚未有工作关注内存池中网络开销大的问题；
10. **计划进度、预期成果 Research schedule, and expected outcomes**

3月到5月：

通过更改内存子系统实现客户机与宿主机的内存统一分配，形成内存池架构；

6月到9月：

测试内存池的性能瓶颈，并设计实现相应的优化方案；

10月到11月：

撰写论文，准备答辩；

1. **与本课题有关的工作积累、****已有的研究工作成绩。Prior experience and accomplished achievements related to the proposed topic.**

**有关的工作积累：**

对远程内存系统进行调研，分析了其解决的问题，并得到其中设计方案的可借鉴之处；

**已有的研究工作成绩：**

巨型虚拟机的微观、宏观性能测试；

巨型虚拟机的内存亲和性线程调度策略；

**本人承诺：开题报告中的内容真实无误，若有不实，愿承担相应的责任和后果。****I hereby declare and confirm that the details provided in this Form are valid and accurate.** **If anything untruthful found, I will bear the corresponding liabilities and consequences.**

**学生签字/Signature of Student： 日期/Date：** 2022-03-21