Report of LegoOS

Summary of major innovations

LegoOS意图打造一个分布式的操作系统，做到以硬件资源级别的划分方式，利用专用的处理硬件和高速网络带宽来实现可扩展性和高性能之间的完美权衡．其中最主要的贡献在于提出新的硬件体系结构来分离处理器和内存之间的功能性依赖关系，并且实现新系统得以在（物理和逻辑上的）分离化硬件资源上运行并实现交互得以给应用提供基本功能．

Problems the paper mentioned

以前的提出的各类单机版本系统架构（无论是Monolithic, Micro, Exo) 都无法很好的解决在多机上的各类问题: 比如资源利用率低下，硬件的可扩展性较低（难以适应动态的添加，减少，改变硬件资源），对于硬件失效的容忍率极低，对于异构体系结构不能很好的支持．而对于已经提出来的分布式操作系统（Multi-Kernel）则依然需要各种OS运行在多个主机之间使用网络通信，有时为了保持一致性会牺牲大量性能，另外他们其实也很难以支持动态扩展或者硬件容错机制．

Related works/papers

高速网卡的发展是最重要的一点，因为对于运行在多个硬件资源之上不可避免的需要以来网络进行通信（制作专用的通信硬件不具有普适性），Infiniband的速度已经能到达内存带宽的一半左右，这使得这类型的网络传输已经相对于内存不再是绝对的瓶颈存在了，甚至在未来，可能会发生网络带宽高于内存带宽的情况，这说明以前的网络远远慢于磁盘远远慢于内存这个硬件假设可能不复存在，从而也会引来新的体系结构的变迁，也使得硬件资源分离通过网络互相通信的方式在合适的处理下可能不再是性能的瓶颈，而可以和单机上的速度具有一定的可比性了．

另一方面，RDMA的机制可以通过网络把资料直接传入计算机的存储区，将数据从一个系统快速移动到远程系统存储器中，而不对操作系统造成任何影响，这样就不需要用到多少计算机的处理功能，也为基于硬件资源的分离，尤其是处理器和内存的分离提供了硬件方面的支持和抽象．也就是说，硬件资源已经可以独立于处理器就能够访问网络这一点为硬件资源分离的体系结构提供了很好的硬件基础．

Intriguing aspects of the paper

同样是尝试解决分布式操作系统的痛点，这篇论文则尝试对硬件资源的分离来入手，也就是对Processor, Memory, Storage每个组成部分作为一个单元，称为pComponent/mComponent/sComponent，而利用一些硬件以及软件辅助来抽象这三组资源，并在上面进行管理，各个组成单元只在需要的时候进行网络通信即可．好处就在于管理一个组成单元的时候，不需要发生对于其他组成单元的局部访问．（做到了组件之间的弱耦合，性能则由RDMA之类的硬件保证）

而且因为处理器与内存，存储之间的割裂，所以需要在内存和存储系统也需要一些能够进行处理的单元，所以会需要一些额外硬件支持．其中设计方面主要有以下亮点：

1. Cache划分给pComponent，这样缓存可以比较好的靠近处理器，而且考虑到程序的局部性，相较于将缓存放到mComponent可以很好的提高整体性能
2. TLB，page tables等技术放到mComponent，这样的好处在与内存管理的部分完全独立于pComponent，也可以使得不同的mComponent可以各自使用不同方式管理自己的内存，当然这

样的分离方式使得Cache都是虚拟Cache了
3. 但是在实际上，再仔细设计和考虑到访存的局部性，pComponent也会拥有额外的小内存，但是这个小内存在逻辑上被看做是分离之后的内存集合的缓存．同时对于这一块小内存会进行重新设计，加上一些额外硬件支持保证了访存的合理性和性能．另外这个小内存ExCache也可以为LegoOS提供一下局部数据（保证不会跨越边界的数据的访问和修改）的存放，而且全部利用物理地址来访问以避免各种TLB等问题．这个Excache的Hit路线有硬件保证，miss路线则交由OS处理．
4. pComponent的线程不会去阻塞等待网络中断的数据，而是以忙等待的方式因为Infiniband太快了，阻塞不值得．同时由于不需要考虑内存分配的问题，对于pComponent而言调度的目的成为最小化切换开销而非最大化利用率．
5. 只有内核进程跑在mComponent上面，所以这个上面的处理器质量不需要太好，只要不成为内存和网络的瓶颈就好．同时对于内存的处理想法也比较基本，就是类似采取master进行分配请求相应并要求全局调度器分配slave上的内容，而分配完成之后由slave完成对应的访存服务的一个想法．一个优化就是对于那些.bss或者匿名内存之类的段延迟到写的时候再在mComponent分配具体实际内存对应，在此之前由exCache管理．
6. sComponent的设计也非常的不太相同，因为为了界面的分离则需要传送更多额外的信息，就是总是通过完整的文件名和路径进行pComponent和sComponent之间的通信，当然其实可以由extended Linux state layer来完成fd和完整路径的对应关系．另外为了使用mComponent作为缓存，整个请求路径变为pComponent先找extended Linux state layer查询在哪个mComponent（利用Hash)，然后mComponent或者服务或者找sComponent提供服务
7. 然后毕竟还是需要全局的各种资源的协调和管理，类似其他的数据中心的服务模式以及性能的考虑，也会使用专门的节点来做为类master的内容进行服务全局资源的申请调度请求，同时其实只需要维护大致的资源利用程度即可，所以不需要每时每刻进行信息采集浪费大量资源和降低性能，只需要每隔一段时间进行heartbeat和资源剩余询问即可．这里比较神奇和简化的是对于一个vNode（暴露给用户的一个虚拟机资源）的所有文件内容只放在一个同一目录下，并且放在一个sComponent，理由是datacenter applications基本上只用到不太多的文件
8. 冗杂提供的是内存级别，这个很合理，因为存储级别的基本上目前数据中心已经研究的很透彻了，而进程级别的冗杂在这里也没意义，因为进程不能跨机器．实现方式也就是通过日志备份在另外的mComponent上，并且需要两个同时完成应答，但由于并行性和拓扑设计，付出的额外性能代价很小，当然同时还要刷一下日志到sComponeng，并且为了性能，对于备份的态度就是性能至上，所以备份影响到性能就不备份了，这时候崩溃就报错即可．
9. 另外，最值得一提的就是整个设计其实是基于datacenter的引用场景下提到的，整个OS其实意图提供的不是一个可以跨越所有硬件资源的然后给用户一个是完整机器的影响，而是提供的是进程级别的隔离，也就是类似虚拟机的思路．也就是说，一个多线程程序其实是无法很好通过这个OS完成所需要的功能的，最多也就只能在一台pComponent上不同的核心跑．而进程之间的通信方式也就只能通过massage passing来完成．

Experiments: test/compare/analyze

一如既往的，使用micro-和macro-benchmark来进行测评，大致结果如下：

1. LegoOS为了Infiniband，RDMA实现的网络栈效率很高，能够轻松超过Linux的本地网络的网络栈
2. 内存性能在优化之后，虽然没有超过LINUX，但是也能够有一较之力
3. 存储方面（使用的是SSD）实际上也比Linux差一截，只有在RandRead的时候差的不多，主要原因是RandRead的时候网络就不再是一个瓶颈．不过这个结果还是相对可观的，如果考虑使用HDD的话，至少存储方面，网络在顺序读写也不算一个瓶颈，所以带来的额外性能可能相比之下很小．
4. 合适的调整Excache，可以使得LegoOS的性能相比于Linux不会太差，2x以内．
5. 当然他也和其他的存储在其他逻辑节点（交换文件）的解决方案进行了对比，性能比较好．（这里其实有些不太理解交换文件的工作方式和在此处对比的意义？）

Places the reseach can be improved

实际上这个不支持Coherence，（这里的Coherence应该更倾向于指代Cache Coherence，对内存的同一个位置的多个机器的可见顺序性）．实际上又是将内存模型的使用推给了应用人员，给予应用人员更复杂的心智负担．不够也不得不承认维护一致性这个事情对性能应该是会带来极大的影响．

另外其实还存在这Consistency的问题，这个问题的确也值得进行进一步的解决，然而这个更多是体系结构的问题，操作系统也应该考虑到这些问题，如果可能可以考虑提供合适的Consistency的级别的API以便用户进行进一步的要求．

其实如果可能的话，也可以考虑为了性能牺牲Linux ABI的，但是的确这样的话故事就不好讲了，但是有时候向前兼容不一定能带来更好的性能．

另外，怎么说呢，就感觉有一种把虚拟机做到了OS级别的味道在里面，加以实现了一些常用的大数据平台的管理调度方式，OS的最大的contribution应该还是他实现了这个东西吧．

What I would do if i wrote this paper

因为之前其实在本科阶段也有想过实现一个支持分布式的操作系统，甚至也和操作系统课程到时以及其他导师讨论商量过，当时是想做在虚拟化那一个层面的，就是利用虚拟化的手段给上层操作系统提供一个完整的内存接口等等，并利用Page Fault来进行获取外面的页面，然后实现透明化，然而不得不说Page Fault的开销在当时看来其实还是很大的，当时也考虑过RDMA的事情，也考虑利用本身内存页面的换入换出算法然后得以实现Swap的功能，从而实现大页面的网络功能。然而当时其实还更多的是考虑需要实现其一致性问题，就是如何保证在不同机器的多线程对于同一个逻辑内存地址的访问，并且甚至考虑能都将利用更底层的比如放宽一致性模型来更进一步的提升效率。当然最后其实感觉可行性和创新性其实并没有想象的那么高，另外其实虚拟化没那么必要，但是直接修改内核也有点困难

Survey Paper Lists

[1] Mellanox. ConnectX-6 Single/Dual-Port Adapter sup porting 200Gb/s with VPI.

[2] Mellanox. Rdma aware networks programming user manual.

[3] K. Elphinstone and G. Heiser. From l3 to sel4 what have we learnt in 20 years of l4 microkernels?

[4] D. R. Engler, M. F. Kaashoek, and J. O'Toole, Jr. Exokernel: An operating system architecture for application-level resource management.