Disaggregated Operating System

Yizhou Shan, Sumukh Hallymysore, Yutong Huang, Yilun Chen, Yiying Zhang
Purdue University

Recently, there is an emerging trend to move towards a *disaggregated* hardware architecture that breaks monolithic servers into independent hardware components that are connected to a fast, scalable network [1, 2, 3, 4, 5] components. The disaggregated architecture offers several benefits over traditional *monolithic* server model, including better resource utilization, ease of hardware deployment, and support for heterogeneity. Our vision of the future disaggregated architecture is that each component will have its own controller to manage its hardware and can communicate with other components through a fast network.

OSes built for monolithic computers can not handle the distributed nature of disaggregated hardware components. Datacenter distributed systems are built for managing clusters of monolithic computers, not individual hardware components. When traditional OS operations spread across hardware components over the network, these distributed systems fall short. Clearly, we need a new operating system for the disaggregated datacenter architecture.

We propose the concept of *disaggregated operating system* for the disaggregated datacenter architecture. The basic idea is simple: *When hardware is disaggregated, the operating system should be also.* There are three main challenges in building a disaggregated OS:
1) How to cleanly separate OS services and map them to hardware components? Traditional OS services are tightly coupled, which assume all hardware resources are accessible within one machine entity. Furthurmore, different hardware components have different constraints. For example, memory components will have plenty of memory but has limited processing power, usually just a memory controller, while processors will have limited amount of memory.
2) How to ensure failure independence? We envision the scale of disaggregated architecture to have at least thousands of components. With this scale, failure is inevitable. Since an application running on disaggregated architecture can be using a set of hardware components and one component can host multiple applications, a component failure should not affect the running state of applications.
3) How to support existing datacenter applications? All current datacenter applications are designed to run on monolithic servers. To provide complete transparency and backward compatibility, we need to hide the resource disaggregation nature in a disaggregated OS. However, doing so can lead to reduced flexibility in resource management.

We are building LegoOS, a distributed, loosely-coupled, failure-independent OS, designed and built from scratch for disaggregated hardware architecture. LegoOS runs a *component manager* at each hardware component and appears to applications as a set of distributed servers. It consists of three types of component managers: processor manager, memory manager, and storage manager. These components can be heterogeneous and can be added, restarted, or reconfigured dynamically without affecting the rest of the disaggregated system. LegoOS uses coarse-grained, global resource management to allocate, schedule, and coordinate across components.

LegoOS cleanly separates the functionalities of different component managers, with zero or only minimal dependencies across them. This clean separation not only makes it easy to deploy, add, and remove components but is also essential to ensuring failure independence. Our approach is to build each LegoOS manager as a *stateless* service and managers communicate with requests that contain all the information needed to fulfill them. For example, our storage manager is built in the NFS stateless server style [6]. LegoOS processor and memory managers are also cleanly separated in that memory managers manage all physical memory and virtual memory addresses, their allocation, deallocation, and mappings, and processor only views virtual memory addresses assigned by memory managers.

LegoOS uses a combination of process checkpointing and memory replication to handle component failure. Different from traditional process checkpointing with the monolithic server model, there is no local storage attached to processor or memory, making checkpointing persistent more costly. Thus, instead of always checkpointing to storage, we replicate checkpoints in memory components and only make them persistent when memory space is scarse. Replication in memory also has the benefits of sustain memory component failure in addition to processor failure.

To achieve backward compatability, LegoOS supports common POSIX APIs and reuses the Linux syscall interface. We also port several drivers from Linux and implement both a customized, efficient network layer and the TCP/IP network stack.

To evaluate LegoOS, we emulate hardware components using commodity monolithic x86 servers. For example, to emulate a memory component, we only enable one or two cores of a server, while to emulate a processor component, we limit the accessible physical memory of a server and use it as last-level software managed cache. We are using InfiniBand as our high-speed network connection. We expect to have the initial results of LegoOS before the SoCC conference.

References

[1] K. Asanovi. FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers, February 2014. Keynote talk at FAST '14.

[2] P. Costa, H. Ballani, K. Razavi, and I. Kash. R2c2: A network stack for rack-scale computers. In *Proceedings of the ACM SIGCOMM 2015 Conference on SIGCOMM (SIGCOMM '15)*, London, U.K., August 2015.

[3] P. Faraboschi, K. Keeton, T. Marsland, and D. Milojicic. Beyond processor-centric operating systems. In *15th Workshop on Hot Topics in Operating Systems (HotOS '15)*, Kartause Ittingen, Switzerland, May 2015.

[4] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker. Network requirements for resource disaggregation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 249–264, GA, 2016. USENIX Association.

[5] S. Legtchenko, H. Williams, K. Razavi, A. Donnelly, R. Black, A. Douglas, N. Cheriere, D. Fryer, K. Mast, A. D. Brown, A. Klimovic, A. Slowey, and A. Rowstron. Understanding rack-scale disaggregated storage. In *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*, Santa Clara, CA, 2017.

[6] R. Sandberg, D. Golgberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the sun network filesystem. Artech House, Inc., Norwood, MA, USA, 1988.