**DSC-204A: Scalable Data Systems, Winter 2024**

# Guest Lecture - 02/21/2024

*Lecturer: Hao Zhang* *Scribe: Ritwick Manatkar, Aseem Dandgaval*

Today's lecture is a guest lecture by Prof. Yiying Zhang about the vision for the future of Cloud Computing. Prof. Yiying Zhang is an Associate Professor at the Computer Science and Engineering Department at UC San Diego specializing in Operating Systems, Distributed Systems, Architecture, and Machine Learning Systems. The lecture is focused on the topic of building Disaggregated Data Centers.

## 1 Introduction: Towards a Fully Disaggregated Data Center

This is a topic that the research group of Prof. Yiying Zhang has been working on for the past 5-6 years. Additionally, this is also an important topic currently being studied in the data center and cloud computing domain.

### 1.1 Overview:

The cloud architecture traditionally over the years has been similar to a personal computer. This means that the server is similar to a personal computer with an OS/Hypervisor on top of it. Additionally on top of this services are provided Like S3, EC2 etc. This is seen in Fig. 1.

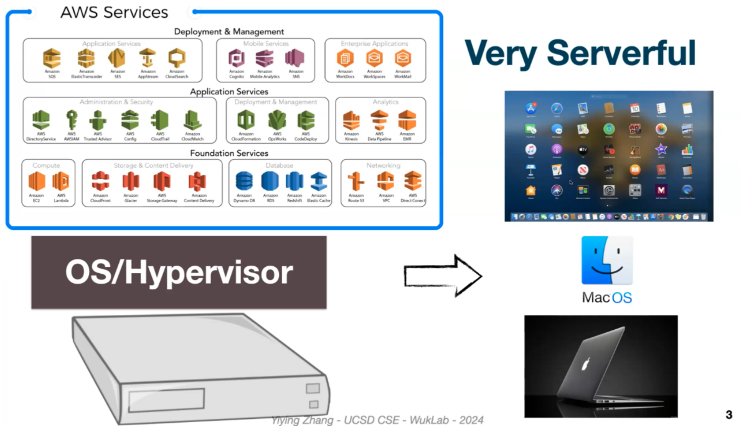


Figure 1: Similarities between Server and Personal Computer Architecture.

#### 1.1.1 What are the disadvantages of this model?

1. Resource Allocation can be a problem, leading to inefficient multi-dimensional bin packing. What is the multi-dimensional bin packing problem? It is the problem of packing multiple dimensions of resources into a set of machines. An example is seen in Fig. 2 where even though the machines have enough CPU

resources(seen in yellow), memory resources(seen in blue), and disk space(seen in red) the waiting job cannot be run as the resources need to be on the same machine. This problem has led to severe resource underutilization by data centers and cloud service providers like Google and Alibaba, whose CPU/Memory usage lies in the 20-60% range.

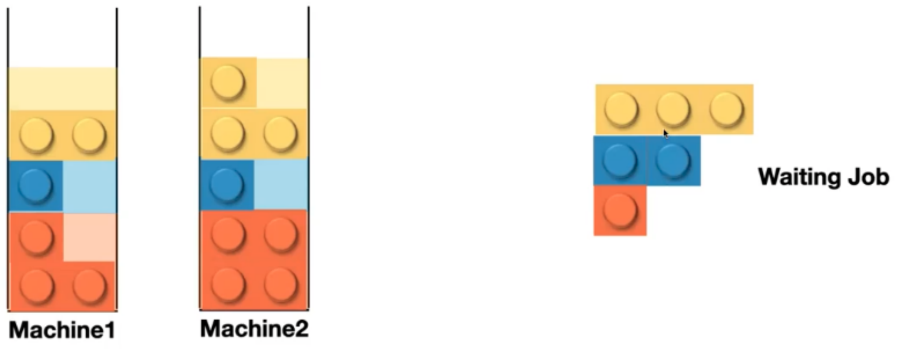


Figure 2: Example of inefficient multi-dimensional bin packing of resources

2. Inefficient scaling of resources is another problem faced by this model. Assume, there is a data center that monitors their resources and finds that they require additional CPU resources to cope with their workload. Then, the data center has to buy a whole machine but only use the CPU of it. This shows that the resources cannot scale independently(seen in Fig. 3).

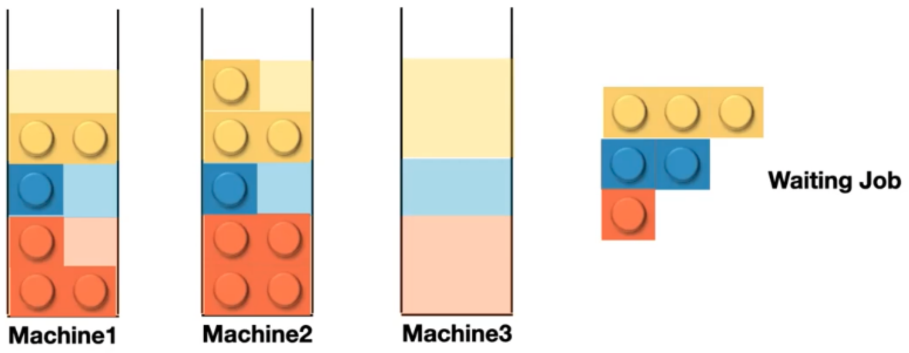


Figure 3: Inability of resources to scale independently

3. Difficulties are faced while incorporating heterogenous hardware components. Over the years, different hardware technologies like GPU, TPU, FPGA, SmartNIC, and NVM have been developed which have required their own PCI/memory slots to plug in these devices. If slots are unavailable then new servers have to be bought. Additionally, the software that runs on this hardware needs to be integrated with these new hardware devices which requires installation and management tasks.

### 1.1.2 What are some of the current solutions?

One of the popular solutions is the software architecture called Microservices where applications are broken into smaller units. These smaller units can be deployed on different machines and they utilize network communication to interact with each other.(seen in Fig. 4).

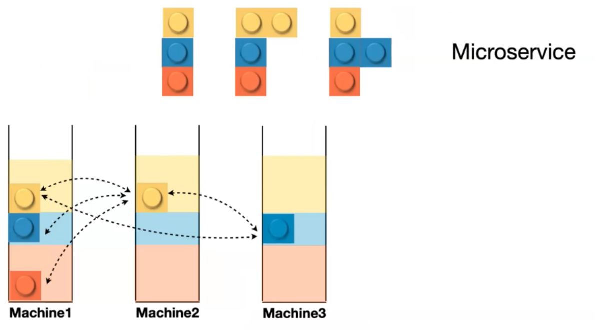


Figure 4: Microservice Architecure

This solution currently poses certain problems:

- It does not solve heterogeneous hardware infrastructure problems.
- It does not allow independent scaling of the system.
- It is hard to manage in terms of multi-dimensional bin packing and network communication due to current software and hardware stack setups.
- There is a lot of room for improvement in terms of performance.

### 1.1.3 Proposed Solution: Hardware Resource Disaggregation

Prof.Yiying Zhang's group has proposed a solution to the aforementioned problems called Hardware Resource Disaggregation which is done by breaking monolithic servers into distributed network-attached hardware components. The core idea is that the CPU, memory, and disk that build up a traditional server are broken down into individual components and connected directly to the network.

The utilization of this can be done by forming memory pools, CPU pools, GPU pools, etc. and resources can be allocated to any individual device in the resource pool. This provides the data centers with numerous advantages like:

- Rightsizing: Use exactly the required amount of resources and the rest is turned off.
- Scale freely: Resources can be added on a requirement basis.
- Low cost of operation: Due to rightsizing and free scaling, costs of operation drop.
- Independent management: Each resource can be easily and independently managed.

Prof. Yiying Zhang shares her vision for the next generation fo disaggregated data centers. As seen in Fig. 5, these data centers would have distributed and disaggregated hardware supported by Clio, distributed and disaggregated system software supported by LegoOS, customized compiling done via Mira, distributed and disaggregated applications supported by Zenix, and finally customized network support through SuperNIC.

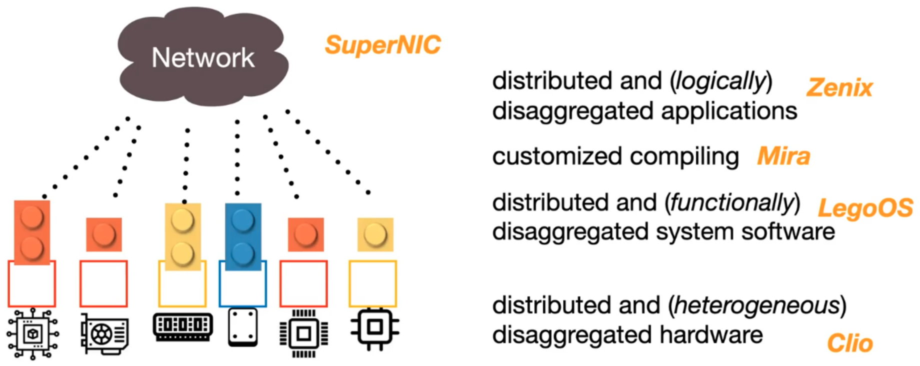

Network
SuperNIC
distributed and (*logically*) disaggregated applications
Zenix
customized compiling
Mira
distributed and (*functionally*) disaggregated system software
LegoOS
distributed and (*heterogeneous*) disaggregated hardware
Clio

Figure 5: Vision for the next generation of disaggregated data center

#### 1.1.4 Audience question: Will networking slow down the performance?

Prof. Zhang acknowledged that networking is the major bottleneck which is solved by providing cross-layer support to operations. The next-generation data centers would have support for the hardware layer, network layer, programming layer, and compiler layer to solve the network bottleneck.

## 2 OS for disaggregation: LegoOS

### 2.1 How to manage disaggregated hardware?

Since existing OS's assume local access to resources, cannot manage distributed resources, and do not have fine-grained failure handling they would not be able to manage the underlying hardware concept. The idea here is disaggregate the resources like process management, virtual memory, file systems, and network stack from each other. As shown in Fig. 6, we can see that this done by attaching a network stack to each unit so that they can be disaggregated. The reason for doing this is that the management becomes a local phenomenon and it avoids crossing the network which makes it faster. This also means that each OS stack can be customized for a device making it easy to manage.

### 2.2 Splitkernel Architecture:

All functionalities are split into monitors. A monitor looks after the functionality for example, a Process monitor monitors the CPU(Processor). Simple messaging is employed between monitors via a fast network like Ethernet or InfiniBand. This is done to avoid the overhead of coherence and focus on explicit application/compiler-controlled messaging. This would also mean that the components are in charge of enforcing coherence. Additionally, the architecture also allows distributed resource management of failure handling. See Fig. 7

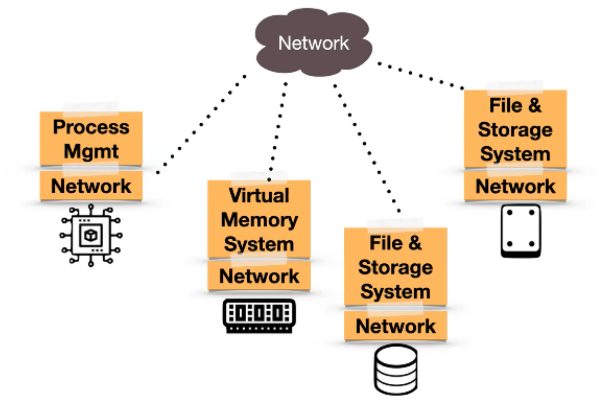


Figure 6: Disaggregated functionalities connected by network

#### 2.2.1 Audience question: How is the networking customized to realize this?

Prof. Zhang's answer: The default method in LegoOS with no customization performs decently but the customization done in Clio performs much better. An even better method would be discussed while discussing the Networking section.

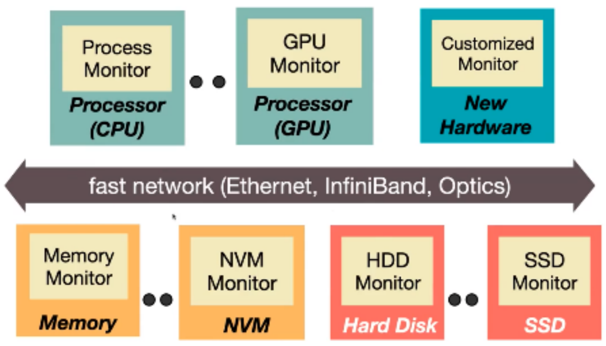


Figure 7: Splitkernel Architecture

### 2.3 How does LegoOS appear to users?

The abstraction between the operating system and applications in LegoOS is done through Virtual Nodes (Vnodes). These act as lightweight VMs or powerful containers, representing sets of Linux-compatible processes for different devices. Vnodes define the production domain, more like containers internally rather than an operating system. One device can have multiple Vnodes as seen in Fig. 8.

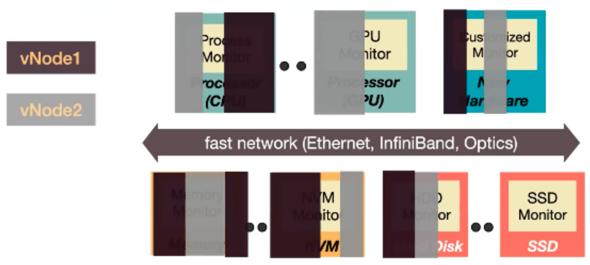


Figure 8: Virtual Nodes in LegoOS

## 2.4 The LegoOS Design

### 2.4.1 Separation of Memory and Processing components

In this lecture, the focus lies solely on the clean separation of OS and hardware functionalities. The main challenge was to separate process and memory, this was done by disaggregating the DRAM and the hardware units that manage memory like the MMU, page table, local site buffer, etc. The virtual memory system is also moved to this memory unit. This results in the remainder consisting of the CPU, the CPU caches, and the Last-Level Cache. The two units are now called the mComponent(memory) and the pComponent(process). This is shown in Fig. 9

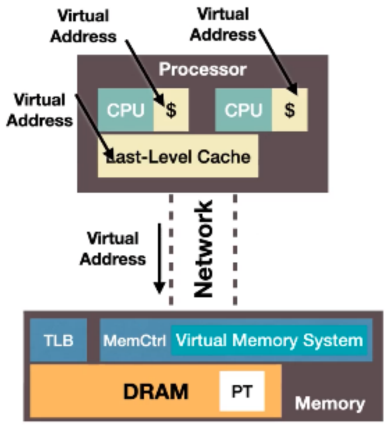


Figure 9: Separation of Processing and Memory Components

This leads to every address provided by the memory component being a virtual address and thus, the CPU is already using virtual addresses in its caches. This means all memory operations happen at the mComponent and a clean separation is achieved.

### 2.4.2 Improvements made for Latency

One problem that this design encounters is when there is a Last Level Cache miss then there is a need to go across the network which introduces latency. Specifically, it is an operation that is 10x slower than using the Memory Bus. This is solved by using an extended cache in the pComponent which is around a few GBs in size. This in turn also means that the DRAM capacity can also be increased(seen in Fig. 10

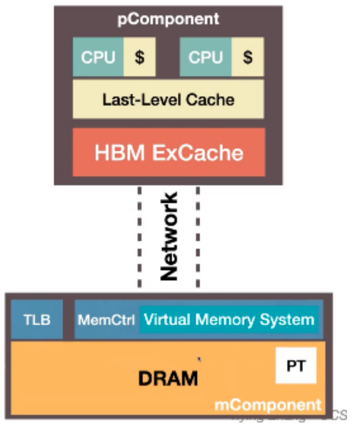


Figure 10: Added Cache in pComponent and expanded DRAM in mComponent

## 2.5 Evalation

A non-disaggregated LegoOS(i.e. everything is running locally) is compared in performance against a Linux swap to local ramdisk system and an Infiniswap system. The performance metric used for comparison is performance/dollar(Perf/$). The results when run on the CIFAR-10 workload are as shown in Fig. 11

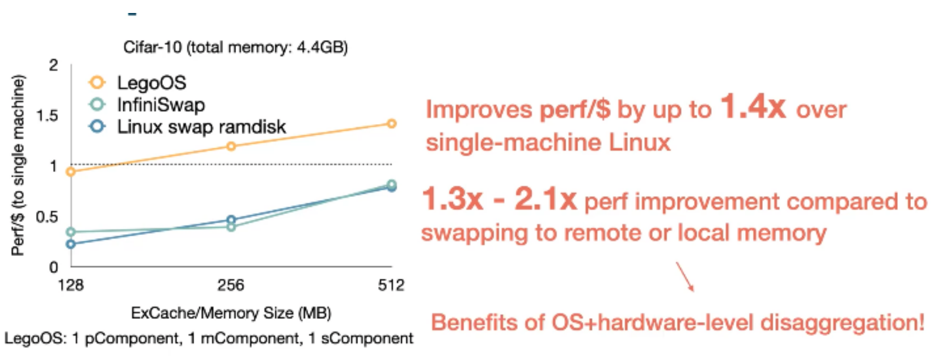


Figure 11: Performance comparison

# 3 Hardware for disaggregation: Clio

Around the publication of LegoOS, there was no real implementation of disaggregated hardware. It was all emulation. Prof Yiyings's team built the first disaggregated memory hardware to connect directly to the network and support multiple processes/applications. The main idea is to eliminate state from the hardware which would reduce cost improve performance, reduce tail latency, and improve scalability.

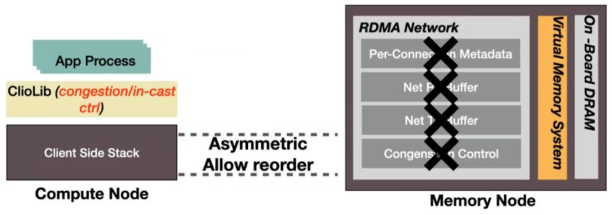


Figure 12: RDMA Hardware architecture

One way to remove states is a customized network. In a traditional system, there is more performance overhead and hardware cost. If you look at memory devices, they are more passive, unlike servers that can't initiate requests, they are response only. If it does not initiate anything, that means that the network is asymmetric, which means you don't need a transmission buffer or a connection layer. The connection info can be merged in the request. Due to the nature of memory access, you don't need a receiving buffer, also because it is one-sided, the computer node knows what requests are going out and what are the responses. So the computer side can do congestion control which eliminates that part as well. As a result, you don't need to run any transport layer which is the main overhead of internet protocols like TCP. So you need a very thin network layer.

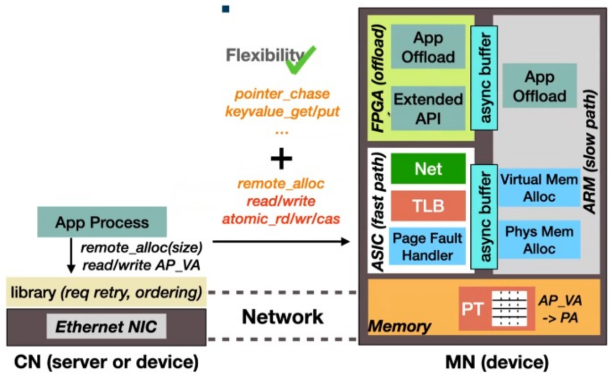
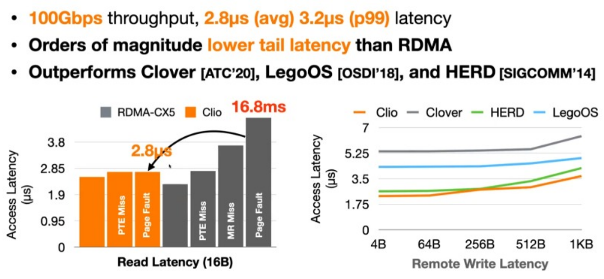


Figure 13: Clio architecture

Clio makes sure that the access fast has high throughput and low tail latency. For more flexibility, Clio also has an offloading pass that allows applications to run FPGA and software. Clio improves performance massively in 3 orders of magnitude when compared to RDMA. In summary, Clio is the first hardware implementation of resource disaggregation which also has very good performance even when compared to commercially available devices.



- 100Gbps throughput, 2.8µs (avg) 3.2µs (p99) latency
- Orders of magnitude lower tail latency than RDMA
- Outperforms Clover [ATC'20], LegoOS [OSDI'18], and HERD [SIGCOMM'14]

Figure 14: Clio performance

## 4 PL for disaggregation: Mira

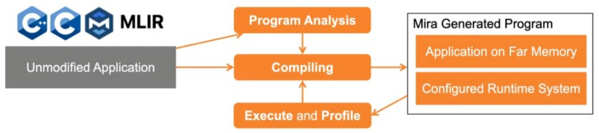
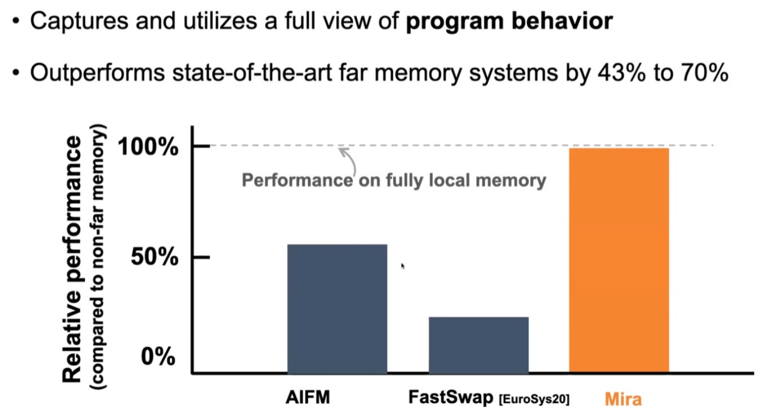


Figure 15: Mira architecture

The existing approach was to build a programming model layer that asked users to re-write their programs to use disaggregated memory or use the transparent layer system like the OS layer. But the problem with this is that programs need to be re-written. Only by doing this, they can explicitly define how they want to use the memory, but this is a big task. On the systems side, adaptation to the application behavior is not possible, so the performance is not very good.

The new proposal was a new middle layer that is concerned with program analysis, compiling, and profiling. If you do everything at the compiler layer, it means no programming change, making it completely transparent. With program analysis and profiling you can observe application behavior and optimize for that in the compiler.



- Captures and utilizes a full view of **program behavior**
- Outperforms state-of-the-art far memory systems by 43% to 70%

Figure 16: Mira compared to state of the art approaches (Higher is better)

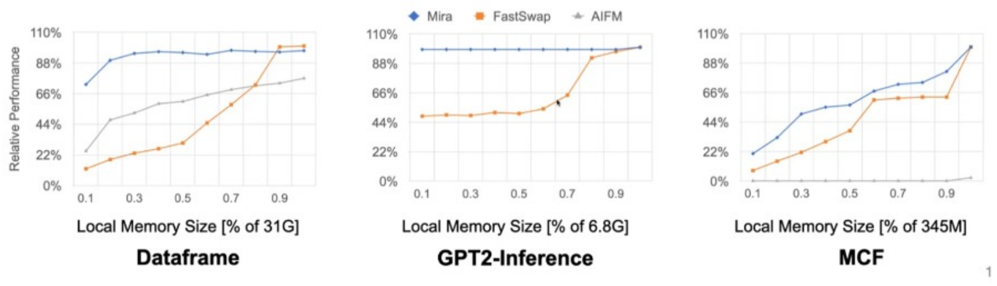


Figure 17: Mira performance across different tasks with SOTA comparison (Higher is better)

Mira does a generic program analysis and compiler framework and automatically figures out how to do perfect pre-caching. In some workloads, it works very well and outperforms state-of-the-art methods.

## 5 Network disaggregation: SuperNIC

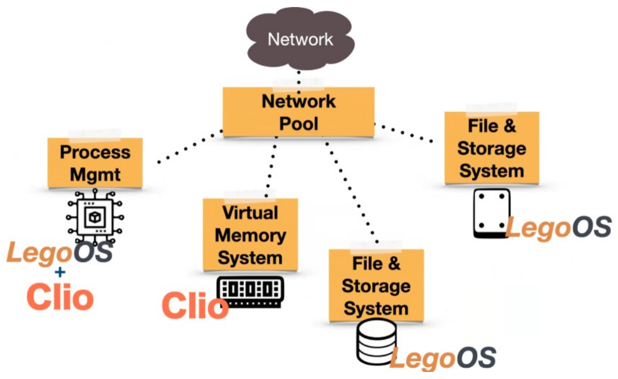


Figure 18: Network Pooling using SuperNIC

Even though Clio reduces the network layer to only the physics and MAC layer, we still need a network stack. Should the network also be looked at as a hardware resource like done previously like disks and processors? Should the network be disaggregated and what will be the benefit of that? Each device needs a basic network connection, and if you disaggregate and consolidate that, we can support heterogeneous load traffic and heterogeneous network functionalities. This is the idea behind SuperNIC (pooling of network resources). SuperNIC does not increase the basic network performance, it does reduce costs and improves performance per dollar.

## 6 Serverless on disaggregation: Zenix

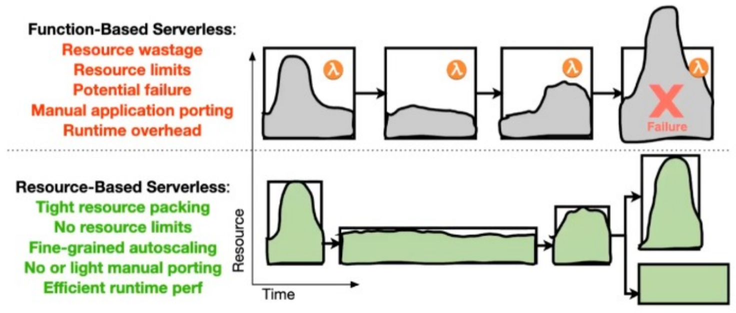


Figure 19: Function-based serverless vs. Resource-based serverless

One of the most popular services on the cloud is serverless computing which promises that you don't need to manage servers, OSes, or any infrastructure. It automatically scales to what you need and you pay for what you use. FaaS (function as a service) is also very popular today. In FaaS the applications are now functions and ideally, you want the resources to be fully used when running these functions, but thats not the case.

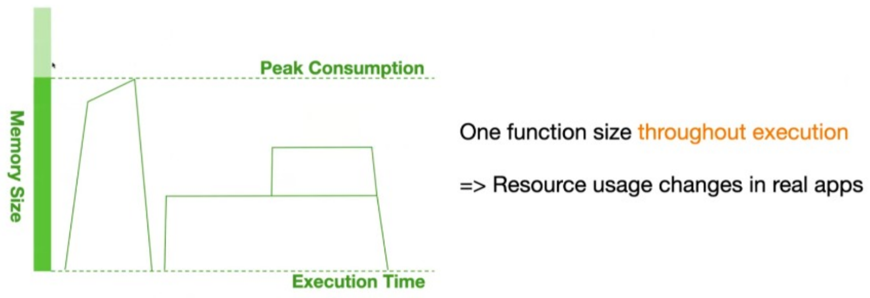


Figure 20: Changes in resource consumption in execution

That is because functions are fixed boxes. You end up wasting a lot of resources. When you call a function and it exceeds the box limits, then the function fails. Even during the execution of a function, the resource utilization is not equal. These challenges are quite difficult to tackle. It is difficult to figure out exactly how many resources to give to each function.

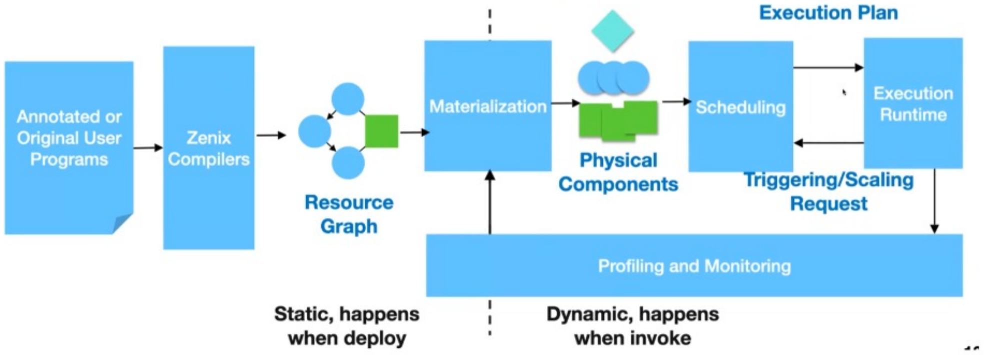
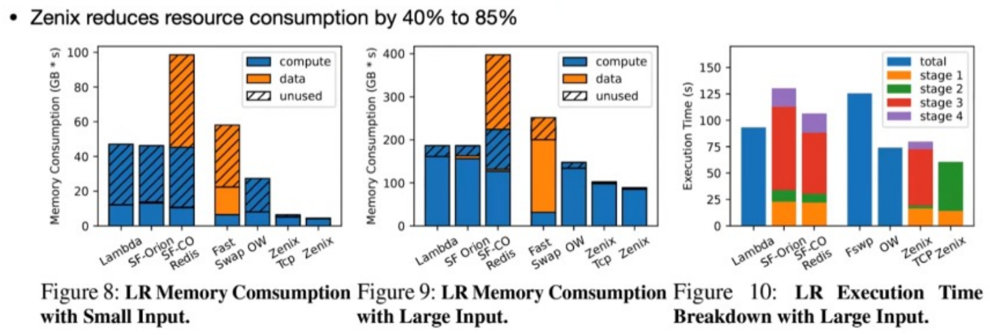


Figure 21: Zenix architecture

A proposal here is that execution should be adapted towards the application instead of users having to re-write their programs to fit the definitions of the function calls and their size. Even if you do split your application in the best way, there still is some wastage of resources. The idea behind zenix is that you should decompose and execute applications based on their resource needs. This can be done using resource graphs.



- Zenix reduces resource consumption by 40% to 85%

Figure 8: **LR Memory Comsumption with Small Input.** Figure 9: **LR Memory Comsumption with Large Input.** Figure 10: **LR Execution Time Breakdown with Large Input.**

Figure 22: Zenix resource consumption compared to SOTA approaches (Lower is better)