Less Boot is better than cold: Scaling out by scaling up

Orestis Lagkas Nikolos
olagkas@cslab.ece.ntua.gr
National Technical Univ. of Athens

Georgios Goumas
goumas@cslab.ece.ntua.gr
National Technical Univ. of Athens

Nectarios Koziris
nkoziris@cslab.ece.ntua.gr
National Technical Univ. of Athens

Abstract

The Serverless paradigm has shifted the interest of the systems community to new techniques that optimize the end-to-end latency per request. Lightweight VMMs or techniques such as snapshots have been introduced in order to reduce the boot time when scaling out new functions. In this poster we take a step back to rethink how the present fundamental approaches affect performance and resource utilization based on the workload nature. We focus on fine managing existing virtual resources rather than optimizing their fast destruction and recreation.

1 Function Execution Models

Each function instance is executed inside a container running on a VM. Cloud providers follow different isolation models [3] defining a concurrency factor (CF) which describes the number of function instances per VM.

Single instance model (CF=1) per microVM: Guest resources are exclusive to one function instance with no other competition. The performance is predictable - differs only between cold and warm invocations. Scheduling new microVMs to servers is like a bin-packing problem where n flavors (CPU/RAM) need to optimally fit in k servers. The overall footprint grows as each instance requires a separate kernel, runtime and initialization code.

Multiple instances model (CF>1) per VM: Resources are shared among concurrent instances and the competition is regulated through OS mechanisms (cgroups). The overall performance is less predicable and depends on the CF and the function workload type. The VMs are bigger than microVMs in order to fit multiple instances, hence increasing the amount of claimed but potentially idle resources. Scheduling bigger VMs is more complicated and leads to free but unusable resources (server fragmentation). However the sum of all instances footprint is reduced by the order of the CF.

2 Challenges & Design

Boot overhead: The CF affects the number of new VM cold boots that will happen during a burst. The single instance model (CF=1) will lead to more cold boots. As the CF increases the cold boot number decreases, as more function instances spawn inside existing bigger VMs. In a function's execution cycle two **workload phases** can exist consecutively. In a **compute bound** phase resources are consumed while in an **IO bound** phase resources are free during the "blocking" window. In serverless, IO requests are usually served by network **connections reinitialized** on each cold invocation with latencies of 100s ms. In the IO phase the arriving requests are handled by new instances. In the CF=1 case higher IO latencies will lead to more microVM cold boots - taxed with the connection overhead. When CF>1 this IO impact is smaller as new instances spawn faster. **Sharing:** In the single instance model the identical stacks -VMM, OS, runtime, function image- are deployed on multiple physical servers maximizing network traffic for distribution and minimizing potential local sharing. When CF>1 the stack is shared -and the guest page cache- among CF instances.

Based on these challenges, we adopt the multiple instance model by scaling out functions in existing VMs while attacking the server fragmentation by vertically scaling the Guest CPU resources directly from the host using cgroups:

From the guest view each instance is pinned to an exclusive set of vCPUs. From a host view the VM process belongs to its own *threaded domain*[1] cgroup. Each instance's vCPU threads form a child *threaded cgroup*, under the parent domain. The VM domain CPU shares is the sum of all of its children shares. The above hierarchy abstracts the CPU allocation from the virtualization boundary as the host can directly control the CPU time of a guest (function) process. The vCPU thread cgroup guarantees $\Omega(VMShares/N)$ of **predictable** CPU time per pinned instance. Thus, the CPU shares of instances blocked in IO can be used by non blocked instances with $O(VMShares)$ in best case (N-1 instances are in IO phase), maximizing the **resource usage**.

When an instance is terminated its vCPUs are unplugged and the corresponding cgroup on the host is empty. A new instance is spawned by hotpluging its vCPUs, creating its vCPU thread cgroup, growing the parent VM cgroup accordingly and pinning the function to the corresponding CPUs in the guest. The VM grows by the bare minimum (compared to CF>1) and a cold boot is avoided (CF=1). Our early-stage work combines the benefits of the state-of-practice approaches and sets our future research direction in exploring efficient memory hotpluging[2] as well as client (IO) connection sharing between instances in the same VM.

References

[1] Threads. . https://www.kernel.org/doc/Documentation/cgroup-v2.txt

[2] Alexander Fuerst et al. 2022. Memory-Harvesting VMs in Cloud Platforms. In *27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS '22)*. Lausanne, Switzerland, 12 pages. https://doi.org/10.1145/3503222.3507725

[3] Zijun Li et al. 2022. RunD: A Lightweight Secure Container Runtime for High-density Deployment and High-concurrency Startup in Serverless Computing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA.