**CLOUD SYSTEMS : PROJECT PROPOSAL**

Name : Sai Ajay Krishna Paruchuru

**BACKGROUND**

Computational caching is a technique systems can use to manage computational redundancy which leads to a change in the system’s representation of execution that converts computational state into a data object allowing to exposing redundancies. The application model that is particularly amenable to computational caching is serverless functions, wherein high-level scripts are deployed on-demand by a remote FaaS platform.

A serverless function is basically a short term code segment written for a high-level language interpreter so that these functions are run by a remote FaaS platform in response to the applied requests. These also promote application designs that are made out of many short-lived executions, which are deployed as singletons, in sequences, or in parallel and offers a powerful primitive for accessing on-demand computation across arbitrary scales.

The operational shift away from the application developer allows the system to control how functions are deployed with in dedicated containers, virtual machines, process-level encapsulation, or language-level isolation. Booting interpreters and compiling source code can lead to long initialization overheads for function deployments. Therefore, to achieve low-latency start times, the FaaS platform must cache intermediate stages along a function’s invocation lifetime for reuse.

Because of these issues, moving forward to enable fast deployments for many different functions, FaaS platforms has to hold huge amounts of state cached in memory. However, the stateless nature of functions enables the platform to cache the execution environment of a particular function for quick reuse for all future invocations. The requirement to cache many isolated environments in memory, many of which are limited in reuse to single function, presents a isolated challenge that modern operating systems have been slow to address.

**PROBLEM**

The traditional caching techniques operating by FaaS platforms are resource-intensive. Consequently, research systems have demonstrated lightweight methods for computational caching within FaaS platforms, such as deploying execution from images of checkpointed processes or forking execution from existing processes. Fundamentally, these systems illustrate that caching can both shorten execution start times and reduce the memory footprints of cached function instances. In our point of view, any user-level caching approach will be insufficient as a general solution for serverless platforms. For example, forking requires cooperation from within the interpreter, which limits the set of interpreters that can be used. In addition, kernel-managed state required for execution sandboxing cannot be captured at user-level, therefore, extending the path of a deployment.

For reduced start times, functions are deployed from unikernel snapshots, bypassing expensive initialization steps. To reduce the memory footprint of snapshots we apply page-level sharing across the entire software stack that is required to run a function. We demonstrate the effects of our techniques by replacing Linux on the compute node of a FaaS platform architecture. We are able to cache over 50, 000 function instances in memory as opposed to 3, 000 using standard OS techniques.

The sudden request bursts are currently unsupported by this traditional approach which can be achieved by the discussed SEUSS method.

**MOTIVATION**

The main idea is to provide a caching explanation for serverless functions that guides a diverse set of language runtimes, executes functions in solitude, and effectively captures computational state across both the application and system levels. We accomplish these properties by deploying functions from unikernel snapshots.

In SEUSS, function logic is arranged with a language interpreter and library OS into an confined unikernel. The flat address space of the unikernel allows a straightforward method for gathering and caching the entire memory footprint and register state of the function. Function state can be captured in a black-box fashion point during execution, as an in-memory snapshot image.

SEUSS is able to significantly scale down the memory use of functions by applying page-level sharing ubiquitously across the entire software stack that is required to run high-level function. This reduction not only fully recovers the cost of replicating kernel state per function, but also improves over the current techniques of platform function caching. Through our method, anticipatory optimizations are achieved by preemptively warming the internal pathways and data structures of the unikernel stack before capturing a snapshot image. This intuitive technique has the dual benefit of reducing the memory footprint and shortening the start time of execution deployed from a snapshot.

Using SEUSS, we demonstrate support for bursts of requests with little impact on background workloads, whereas the Linux implementation fails to appropriately support either. Our key goal is that modifying the representation of execution enables powerful optimizations to squash redundancy across both application and systems pathways. We assure that the SEUSS method is quite simple and general enough that it can be readily adopted by production-grade systems.