FaaS as an alternative to Microservices[1]

IIT Bombay

Status Report as on September 30th, 2019

This report covers work done since Feb 2019. We had submitted an earlier report that described work till Feb 2019.

Literature Review

Research papers from the following categories were studied and reviewed briefly.

- **FaaS-ification of Applications**: FaaS is a new paradigm that is slowly being adopted for building applications. Despite its advantages like no resource provisioning, automatic scaling and pay-per-use model, people are still sceptical in adopting this model as compared to its counterparts like IaaS. This is due to it still being in a nascent stage and the lack of substantial guidance to adopt the same. We looked at the works that have been done to remove the apprehensions of developers in adopting FaaS. These include tools/processes for the conversion of any application into a FaaS based applications, Serverless design patterns, techniques/optimizations to be used in conjunction with existing FaaS facilities, efforts in developing tool to debug FaaS applications.
- **Benchmarking/comparing FaaS providers**: Multiple Cloud service providers have come up with their own Serverless platforms, each of them having unique architectures which are tailor made to be efficient in a specific field of operation. In the space of commercially available solutions, the first to enter the field was Amazon with AWS Lambda functions (AWS), followed by Google Cloud Functions (GCP), Azure Functions (Azure), IBM Cloud Functions etc. A thorough survey of such architectures is thus warranted to get a better picture of each platform's strengths and weaknesses and scopes of improvement upon them. We looked at papers that aimed at benchmarking a subset of the platforms available- both commercial and open-source.
- **Optimizing Serverless Architecture Design**: The brief survey of the open-source and commercial offerings of Serverless platforms above provide some keen insights into the certain performance issues. One of them being that almost all of the platforms have a large cold start latency. This can be an issue for latency sensitive applications like gaming servers, where response latency can affect user experience. A survey of several works aimed at reducing cold start latency, optimizing system internals, optimizing request pipeline and an alternative approach of Serverless architecture design that uses micro VMs instead of containers to run the function instances was performed.

[1] The project is funded by a CSR grant from IFTAS and is managed via project code DO/2018-IFTA001

One question that came out of these studies was whether we could merge the isolation offered by (lightweight) VMs with the speed of startup of a container to address the issue of cold start latency. Toward this end, we investigated MicroVM environments for serverless setups. Here is a brief report on that study:

- **Hybrid Container Environment for Serverless Workloads**: Serverless workloads traditionally run in Container based environments, i.e., every function instance is essentially run in a container. Even though they are efficient, they lack in security. Numerous attacks like rootjail breaks, DoS attacks on co-tenants have been demonstrated. Containers share their Host OS, so any attack on the host compromises the Container. Different workloads need different levels of security. Information sensitive Workloads demand a higher level of isolation. Such demands cannot be met by vanilla Container environments. MicroVMs provides higher level of Isolation but affects performance in terms of the start-up time. A Hybrid Serverless Framework Solution was built using Knative allowing users to deploy workloads in either container based or MicroVM based environment. Latency sensitive users can deploy workloads as containers and Security Sensitive users can deploy in Micro-VMs with higher levels of isolation
- **Micro-Kernel based VMs:** A major part of the cold start latency comprises of the OS boot time. A general purpose VM takes more than a few seconds to boot, which is unacceptable for Serverless platforms. A solution would be to build a leaner kernel targeted to run serverless workloads. This would include stripping down unnecessary device drivers, power management drivers and other kernel modules. As of now, experimentally an ultra-light weight kernel has been shown to be able to boot at sub 300ms. Even though this is impressive but for most workloads the additional 300ms overhead to the function runtime can be an impediment.
- **In-memory suspended VM pool:** An alternative look into cold start latency for MicroVM environments would be to have a pool of suspended VMs kept in the main memory. When a function workload arrives, a VM from this pool can be resumed to serve the request. VM suspend / resume cycles are typically an order of magnitude faster than in the case of booting a VM from a backing storage. This along with a lean kernel and further bring down the coldstart latency. Experimentally, a very lean kernel has been shown to resume in the range of 40-50ms. However, this brings a different problem. Keeping a VM suspended in memory takes resources and hence it would be imperative to investigate mechanisms to increase VM packing density.
- **Out-of-band VM page merging:** The FaaS model works at an application layer abstraction, i.e. the users do not have access to the underlying OS. This allows service providers to have identical OS images across VMs. An interesting aspect due to this homogeneity of the framework is that it is expected that many of the physical pages would be redundant copies of the same guest kernel and user-space pages. This opens an opportunity to perform out-of-band page merging via a transparent background service. This increases VM packing density and resource consolidation. Indeed, a KSM

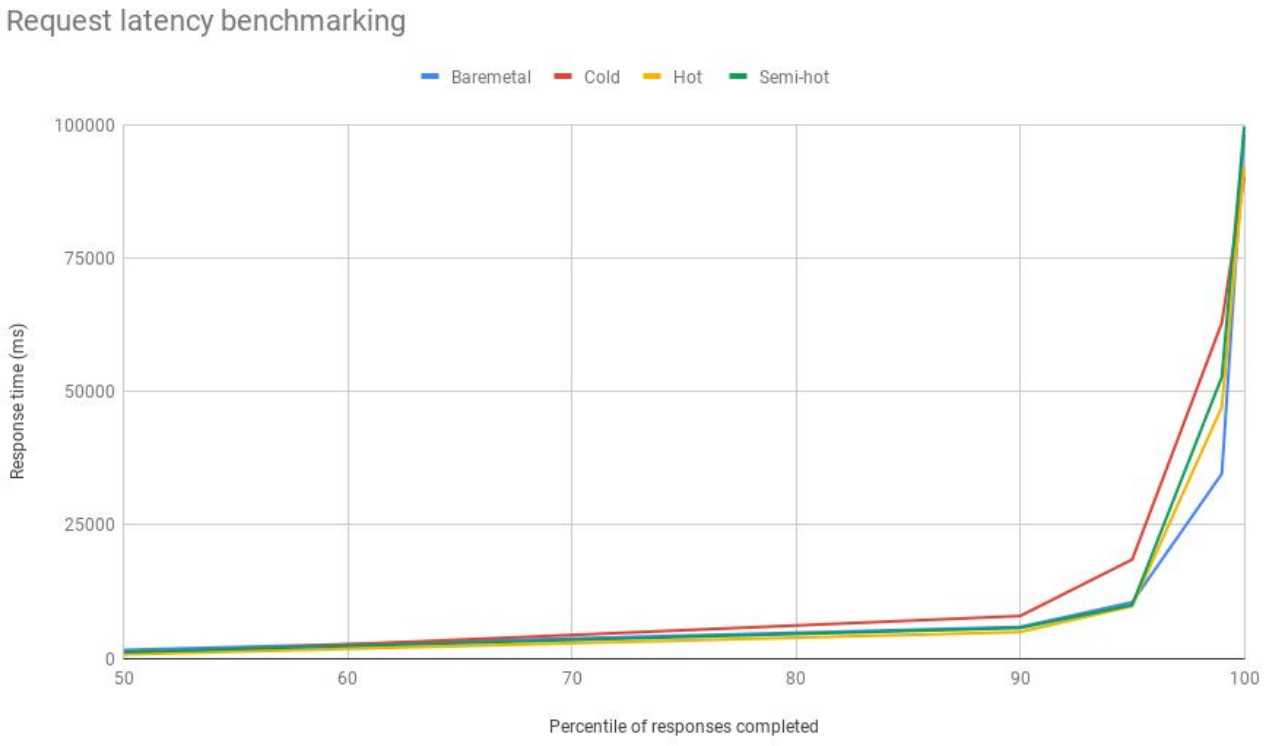
based out of band page merging shows around 15-20% decrease in memory requirements across a testbed of 200 microVMs.

We are currently in the process of looking into these design issues and benchmarking each of these options.

Knative

Knative is a open source environment for serverless function dispatching. It leverages Kubernetes as a resource manager. We intend using Knative as the basis for our own environment and modifying it suitably.

- Studied the various components of Knative architecture: Istio, Eventing and Horizontal Auto Scalar.
- Knative Benchmarking: Prime number series generator (CPU bound function) deployed and tested at 100, 500, 1000, 2000 concurrent requests for a duration of 100s. The benchmarking was performed under different system conditions. This was also compared with bare-metal microservice based performance. The following graph shows the distribution of the response times measured.



Ongoing Work

PanOpticon: A Comprehensive Benchmarking Tool for Serverless Applications

Function-as-a-Service (FaaS) is a new offering in the cloud services domain. Multiple Cloud Service providers have come up with their own implementations of FaaS infrastructure providing end-users with a multitude of choices. Each such platform provides a non-overlapping set of features which satisfies a subset of users. A tool that automates capturing how a function behaves under different configurations of a platform and across platforms will, therefore, be useful for end-users intending to deploy applications as a collection of FaaS units. In spite of the presence of a few benchmarking tools for FaaS offerings, they lack the comprehensive breadth required to understand the performance aspects of the design choices made by the end-users. Most tools focus on tuning resource parameters like memory, CPU requirements and measure metrics like execution time. They lack the option to measure the effects of abstract features like function chaining and choice of function triggers. We have developed PanOpticon - a tool that automates the deployment of FaaS applications on different platforms under a set of tunable configuration choices and presents the users with performance measurements for each configuration and platform.

Based on this work, we have recently submitted a paper to the COMSNETS conference and the paper is currently under peer review. We will share the paper if it gets accepted and acknowledge IFTAS in it.

Future Work

We are planning to look at the following trajectories for MicroVM based Environments for Serverless Frameworks.

- **VM fork / live cloning:** A separate trajectory into the VM packing problem would to have a single in-memory VM base image and perform cloning on the fly, depending on the function workload. This mitigates the need to maintain a large VM pool and since the child VMs are similar to the parent VM, the child VMs can be copied on a copy-on-write basis. This ensures a minimal amount of memory pages must be copied before the child can boot itself minimizing boot-time while maximizing VM packing density.

  However, this would require non-trivial modification of existing VMMs and warrants the development of a more of a FaaS-centric VMM.

- **Effect of Function placement on VM packing density for a multi-host environment:** To investigate the effect of function placement has on VM packing density, when running in a multi-host environment. It can be theorized that if functions running on similar software stacks are co-located the probability of page sharing should increase. This can

be interesting in a multi-host environment where functions running on similar stacks can be placed on the same host. E.g. A node.js based function can be expected to share a larger number of pages with other node.js based functions as compared to a function running on python.

- **A Learning-Based Function placement policy**: Current function placement policies are static and reactive in nature. They scale up and scale down according to the incoming load in a reactive fashion. This heavily penalizes bursty traffic. Function packets carry a wealth of information regarding their use case probability distribution, and workload profile. Secondly, current placement policies do not take into account factors like function co-location and geographical location during function placement. Co-locating functions with similar architecture and software stack (for eg. Python stack) can lead to lower start-up time (due to better caching) and higher resource utilisation (due to better resource multiplexing). It is expected that there might be other hidden features that influences and if tuned correctly can improve the performance of serverless workloads. Feature engineering of such a large feature is not possible and also not suitable. Running traditional ML-based algorithms can also be problematic due to:

    a) Lack of curated dataset
    b) The Malleable and dynamic nature of features in the serverless domain.

  A possible solution can be to run an online reinforcement learning-based algorithm that maximises a reward function based on:

  $R = -- (\gamma_1 L1 + \gamma_2 R2)$,

  where, γ_i is a regularisation factor

  L1is the function latency factor

  R2 is a resource consolidation factor.

  So, the goal of the learning function would be to reduce a composite cost factor comprising of the latency and resource usage, with γ_i acting as a bias factor providing emphasis on latency or resource based on business needs. An interesting exercise can be to design a learning agent that come up with a better scheduling algorithm than what is being used now.