(BSCS 6E) PDC Assignment # 3 L20-0921 Aisha Muhammad Nawaz
Serverless Computation with OpenLambda

Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani†,
Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau
University of Wisconsin, Madison † Unaffiliated

Summary

Open Lambda which is an open-source platform was made to help ease creation of webservices & applications that have serverless computations. This research paper mentions the challenges that might be faced when designing & implementing such systems. Current web applications are compared as well. The Platform was made with the intention to help researchers analyze different approaches to serverless computing & to ease research on Lambda Architectures.

From applications running on their own physical machines, to use of virtualization or containers or even servers, Serverless Computation beats them all. In this model, an application is viewed as a collection of functions instead of servers. Functions have access to a common data store. This style of service construction is also known as Lambda model and has several advantages over server-based techniques such as: lack of stress of server management, inexpensive sending of code & easy scalability. Lambda model has changed the way information is shared between hardware to operating systems to the runtime environments.

Focus is made on lambda environment's implementation, AWS Lambda Cloud Platform. The model helps developers tell functions that shall run in response to different events. In AWS, the cost of an invocation is proportional to the memory cap*actual execution time (rounded to nearest 100ms). Lambda applications are usually used along with cloud database as lambda functions are stateless.

Advantages of Lambda model include quick & automatic scalability. This was proved by comparing AWS Lambda Model to AWS Elastic Beanstalk, a container-based server platform. For one minute the same benchmark ran on both platforms: workload maintains 100 outstanding RPC requests and each RPC handler spins for 200ms. Results were that RPC in Elastic BS took 20s but RPC using AWS Lambda had a median response time of 1.6s. Difference was because all elastic BS requests were served by the same instance. Another advantage of AWS Lambda is that there is no need of configuration for scaling but Elastic BS configuration is rather complex & failed to spin up new workers.

Lambda architecture may be required by future work load & to understand this an analysis was made of client-to-server patterns in a previous RPC-based-application: Google Gmail. Here, RPC Calls made were traced by a chrome extension that injects wrappers. Correlation of RPC traces was made with Chrome's network trace. Workload was of refreshing inbox page to keep browser caches warm. RPC calls corresponded to 32% of all requests & took 92ms median time, longer than other requests with 18ms median time.

There are two types of RPC Requests: very short & very long. Very short requests (under 100ms) took on average 27ms but as latency was only traced on the client side this is an upper bound on actual time for the RPC handler. A very long requests took 231 seconds (93% time of all requests combined).

Requests cost 3.7 times more than if increments of charges are more fine-grained than 100ms which is a design implication. Solution is to do less but longer RPC calls or reduce lambda initialization costs. Applications use RPCs with long lifetime to support server-side pushes these are blocked waiting for updates. If lambda environments do not provide special support for these lambdas, idle handlers will cost the most.

Research problems exist in server-less computing space. AWS Lambda reuses same containers to execute multiple handlers but lambdas are much slower than containers at low request volumes. A similar setup as before but with light weight was used to test this. Latencies were 10 times worse with AWS Lambdas than with Elastic BS. To make AWS Lambda competitive base execution time needs to be improved.

Tradeoffs exists of running lambda in containers. Keeping containers paused results in high memory cost. Memory is the main bottle neck & paused containers have same overhead as running ones. It is a hard tradeoff to keep non-running containers in the paused or stopped states. Research challenge is to reduce memory costs in paused & to reduce the restart costs from stopped.

Recent challenges are discussed for just- in-time compilation, package management, web sessions, data aggregation, monetary cost & portability. Applying techniques like expensive profiling can be hard with lambdas. To make dynamic optimization efficient for lambdas sharing of profiling data may be required among different lambda workers. If users bundle large third-party libraries inside their handlers, it can increase startup latency greatly. Solution is Lazily copying packages or Lambda platform could be package aware but thar would cause new code locality challenges. Lambdas are short-lived & stateless but users expect to have several related interactions with a web application so Lambda must provide a shared view of cookie state across calls from same user. To ensure two-way exchange of data between client & server protocols used are Web Sockets or long polls but these are difficult for lambda as they are based on long-lived TCP connections. Several opportunities exist to integrate lambdas with databases. Supporting change feed abstraction with lambda has challenges similar to ones with long-lived sessions. Change feed batching must be integrated with Lambda state transitions. Lambda compute model introduces new potential consistency boundaries, based on which actor accesses the data. Applications with search queries over large datasets require special lambda support. To support the scatter/gather pattern, multiple lambdas will need to coordinate in a tree structure so it is important to collocate lambdas with the data & can be done by building custom data stores that coordinate with lambdas but diversity of aggregator applications might motivate users to use variety of platforms to preprocess the data hence defining general locality APIs for coordination with a variety of backends may be necessary. Lambda load balancers have to make low-latency decisions by considering factors like session, code and data locality. Recent lambda integrated tools can help developers debug monetary cost. Decomposing monolithic web applications into lambda based micro services is difficult. Web applications frame-works like flask & Django use language annotations which provide great hint to automatic splitting tools that port legacy applications to the lambda model.

All in all, Lambda model is far more elastic & scalable than previous platforms. Open Lambda shall include several subsystems that will coordinate to run Lambda handlers, a local execution engine, a load balancer & a Lambda-aware distributed database.