Serverless Foundations for Elastic Database Systems

Johann Schleier-Smith
UC Berkeley
jssmith@berkeley.edu

In distributed databases, elasticity is a property that describes how quickly a system responds to changes in the workload, to scaling or shifting demand for resources [4]. Such resource elasticity is one of cloud computing's key promised benefits, and a number of transactional, big data, and information retrieval systems have demonstrated its value, typically by scaling by multiples of 2-3x on timescales of minutes to hours.

Recent advances in serverless computing, the implementation of the pioneering AWS Lambda Function as a Service (FaaS) platform [1] preeminent among them, have established expectations of elasticity that are difficult for databases to match. For example, PyWren scales up from zero to hundreds of processors in seconds, and to thousands of processors minutes, drawing from multi-tenant resource pools maintained in the public cloud [3]. With equal ease, it scales down to zero resource consumption. When used in combination with a serverless FaaS system, an elastic database can easily become a bottleneck. We suggest that the design principles behind "stateless" FaaS platforms point the way to system abstractions that offer transformative improvements in the elastic scalability of state-intensive applications.

FaaS was designed to offer users elastic scalability and complete relief from server operations. Users configure *functions*, bits of application code, that run in response to *events*, bits of data on queues. Functions run on compute instances provisioned by the cloud operator, and since function state is ephemeral and function execution is time-bounded, the operator has many opportunities to reclaim resources, to create more of them, or to invoke one of the most time-tested troubleshooting procedures, the instance restart.

FaaS is often described as stateless but in fact most implementations recycle compute instances over many function invocations, with instance state persisting between them. Thus it is natural to ask whether one can build a database system that uses FaaS functions as compute and FaaS ephemeral state as memory, memory for keeping the database buffer pool among other things. The trouble is that FaaS lacks a way of addressing functions; invocations can go to any available instance. In our proposal, we extend FaaS by partitioning the function execution instances across an invocation key space. Put simply, `invoke(func,args)` becomes `invoke(func,args,pkey)`.

As illustrated in Figure 1, the serverless runtime establishes a partitioning on the space of keys, then routes function invocations for each key to the same instance consistently. Additional configuration allows users to specify whether partitions may be served by only one instance, or by many, e.g., to support replicas for read scalability. If a partition becomes overloaded, the partitioned FaaS (pFaaS) runtime terminates the associated function instance, splits its key set, and creates two or more new ones in its place. Merging key sets for scale-in works similarly. Virtual actor systems [2] also build working state from a backing store as needed, so pFaaS could offer a practical route to Actors as a Service (AaaS).

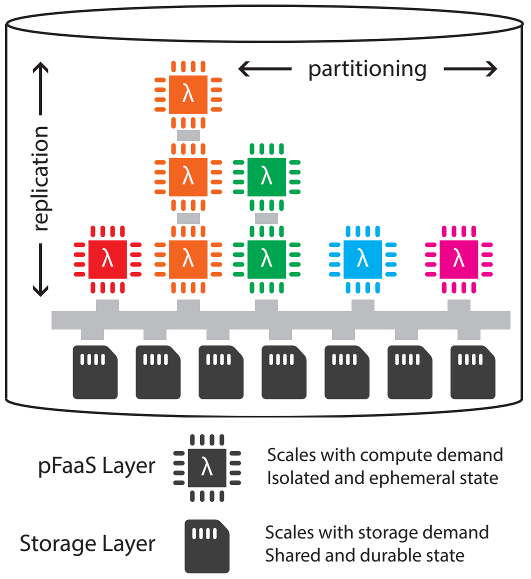
partitioning
replication
λ
pFaaS Layer
Scales with compute demand
Isolated and ephemeral state
Storage Layer
Scales with storage demand
Shared and durable state

Figure 1: An elastic database with a shared disk architecture built from partitioned FaaS (pFaaS) serverless compute and serverless storage (e.g., AWS S3 or EFS). FaaS need not be truly stateless, and we make its cached state addressable and accessible through keyed function invocations.

Several challenges remain before we see databases and other stateful applications built atop pFaaS. Previous scalable databases have used a shared-nothing approach, but achieving high levels of elasticity requires scaling compute, cache, and storage separately. Existing clustered database techniques may also struggle with rapid changes in node count, and with node quiescence between function invocations. Transactional workloads will be more challenging to implement than big data or analytics on account of their demanding coordination and latency requirements. Still, we believe that pFaaS promises to provide a foundation for a broad variety of stateful applications, allowing them to achieve serverless elastic scaling similar to that today enjoyed by stateless FaaS applications.

REFERENCES

[1] [n. d.]. AWS Lambda - Serverless Compute. https://aws.amazon.com/lambda/.
[2] Philip A Bernstein et al. 2014. Orleans: Distributed virtual actors for programmability and scalability. *MSR-TR-2014–41* (2014).
[3] Eric Jonas et al. 2017. Occupy the cloud: Distributed computing for the 99%. In *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 445–451.
[4] Rebecca Yale Taft. 2017. *Elastic database systems.* Ph.D. Dissertation. Massachusetts Institute of Technology.