Narrowing the Gap Between Serverless and its State with Storage Functions

Tian Zhang, Dong Xie, Feifei Li, Ryan Stutsman
Shredder

- A multi-tenant in-memory key-value store.
- Extensible with user-provided storage function.
- **5 M ops/s** per machine, **~20 μs** latency
- In-runtime data access method, able to access **10s of GB** of data per second.
High growth of serverless computing

Why Serverless Computing Is The Fastest-Growing Cloud Services Segment

September 2, 2018
This is the first in our three part series on serverless computing. Check out how the major cloud providers are getting into the space here and take a look at the final installment on early-stage companies to watch here.

The serverless market is expected to reach $7.7B by 2021, up from $1.9B in 2016.

Serverless is a new type of computing model that allows organizations to manage only the computing resources they need for each distinct action.
High growth of serverless computing

Survey Shows More than 75% Use or Plan to Use Serverless in Next 18 Months

- Yes: 50%
- No, but planning to use serverless architecture in the next 18 months: 28%
- No, and not planning to use serverless architecture in the next 18 months: 22%

Source: The New Stack Serverless Survey 2018: Is your organization using a serverless architecture? or 528
High growth of serverless computing

Global Serverless Architecture Market to Reach $21.99 Billion by 2025 at 27.8% CAGR: Allied Market Research

Rapid rise of the app development market along with increase in demand for useful applications for different platforms such as Android and iOS have boosted the growth of the serverless architecture market.
Advantages of serverless computing

- **Fine-grained** resource provisioning.
Advantages of serverless computing

- **Fine-grained** resource provisioning.
- **On-demand** scaling.

![Diagram showing the transition from server to container to function/serverless with requests and time axes.](image)
Problems of serverless computing

- Shipping data to code paradigm.

Diagram:
- Serverless Function
- High latency
- Bandwidth bound
- Data
- Storage Service
Problems of serverless computing

- **Shipping data to code** paradigm.
- User pay for additional **idle** time.

![Diagram showing serverless function, data storage, and high latency issues.](image)
Narrowing the gap

Network costs between servers ~ 50 μs

Data
Narrowing the gap

Network costs between servers

Kernel bypass to reduce latency

~ 20 μs

~ 50 μs
Narrowing the gap

Network costs between servers

Kernel bypass to reduce latency

Push code to data, process isolation cost
Narrowing the gap

Network costs between servers

Kernel bypass to reduce latency

Push code to data, process isolation cost

V8 runtime isolation, boundary crossing cost
Shredder design goals

- **Programmability** - flexibility to implement any custom logic.
- **Isolation** - functions should be safely isolated.
- **High Density and Granularity** - should support thousands of tenants.
- **Performance** - optimize performance as much as possible.
Why JavaScript

- Flexibility of general programming language.
- Easier to implement customized data structures and logics than SQL.
Shredder design

- Embedded **V8** JavaScript runtime to isolate functions.
- Data access through V8 builtins.
- Data store implemented in C++ native code.
- Networking, data management, etc.

![Diagram showing V8 engine and data store with JavaScript and C++ components](image-url)
Problem: runtime exit costs add up

- Data access across boundary from JavaScript to C++.
- Add up to a lot of overhead for functions accessing lots of data.
One step further

- Direct and safe data access from serverless functions.
- Eliminate boundary crossing.
- Leverage V8 JIT compiler.
CSA to eliminate boundary crossing

- Implement data access builtin in **CSA** (CodeStubAssembler), the V8 internal IR.
- Eliminating boundary crossing to C++.
- Runtime can inline CSA to improve performance.
Data store and CSA builtin co-design

- CSA builtin and data store implement the same data lookup logic over shared data.

```c
db_val_t* ht_get(hashtable_t* ht, uint32_t key) {
    ....
}
```

```c++
TF_BUILTIN(HTGet, CodeStubAssembler) {
    ....
}
```

NIC → CSA → Hashtable
Threat Model

- V8 contexts ensure fault isolation and no cross-tenant data access
  - Data is never shared across tenants
- TCB includes store, networking stack, OS, hardware, and V8 runtime

- Speculative execution attacks complicate secrecy
  - Users could craft speculative gadgets
  - Speculative gadgets could transmit restricted state through cache timing side channel
  - Landscape of attacks still evolving; unclear if runtime/compiler will be able to resolve them

- For now, a shared storage server is only safe with some mutual trust
  - Two-level isolation model possible
  - Process per-tenant; different functions in different runtimes
Evaluations

- 2 x 2.4 GHz Xeon with total 16 physical cores.
- 64 GB memory.
- Intel X710 10GbE.
- DPDK for kernel bypass.
Reduce data movements over network

- **Projection**, queries the first 4 bytes of a value.
- Pushing projection to Shredder reduces data movements, compared to baseline which fetches each whole value.
Data intensive functions

- Traverse Facebook social graph.
- Access 10s of GB of data per second.
- Shredder 60X better performance.
- CSA brings 3X performance gain.
Compute intensive functions

- Neural network inference functions.
- Shredder at disadvantage for compute intensive functions.
- Performance gain still possible if reduces enough data movements to offset inefficiency of JS code.
Related works

● Extensible stores:
  ○ Comet: An active distributed key-value store. OSDI 2010.
  ○ Splinter: Bare-Metal Extensions for Multi-Tenant Low-Latency Storage. OSDI 18.

● Serverless state store:
  ○ Pocket: Elastic ephemeral storage for serverless analytics. OSDI 18.
Conclusion

- Gap between functions and persistent states is costly
- Moving functions to storage eliminates some overhead
- Runtimes lower isolations costs, but boundary crossings still add up
- Data-intensive functions benefit from tighter integration of code and data
- Key idea: embed storage access methods within runtime
  - Both storage server and functions can both access data at low cost
- Result: achieves 3X better performance with in-runtime data access.

Thank you!
Backup

Kernel bypass

![Graph showing throughput (M ops/s) vs length of list for different operations: Shredder, Shredder w/o CSA, Remote get/put.](image)

No kernel bypass

![Graph showing throughput (M ops/s) vs length of list for different operations: Shredder, Shredder w/o CSA, Remote get/put.](image)
The V8 engine communicates with the JavaScript and C++ layers through a Data store, which contains data structures and contexts. Each V8::Context interacts with the data store, facilitating seamless integration between the languages.