Pushing Compute to Low-Latency Kernel-bypass Key-value Stores

Tian Zhang, Chinmay Kulkarni, Ryan Stutsman
stutsman@cs.utah.edu

CNS-1566175

Motivation

- Key-value stores with few µs access times
  - RAMCloud, FaRM, MICA
- Everything in expensive DRAM

- Multi-tenancy needed to be practical
- And, get/put data models limit performance

Can we support safe user-level extensions?

Applications

- Push σ, π, γ
- Alternate data models, ADTs; graphs, TAO
- Heavy operations: matrices, GMM
- *Our interest: pointer-chasing for concurrency control metadata*

Requirements/Approaches

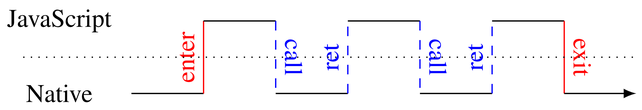
Model	Fast Compile & Install	Fast Runtime In/Out	Isolation	"Pointer Chasing"	Data-Intensive Ops	Compute-Intensive Ops
SQL	✓	✓	✓	w/⋈	✓	Builtins
Native/C++	✘	✘	HW	✘	Hard	✓
JavaScript	✓	✓	✓	✓	✓	✓

- (Near) native performance
- Low invocation overhead
- Runtime reconfigurable
- Inexpensive isolation
- Low installation overhead

Isolation Costs

- Data-intensive procedures mean pressure on protection domain boundaries

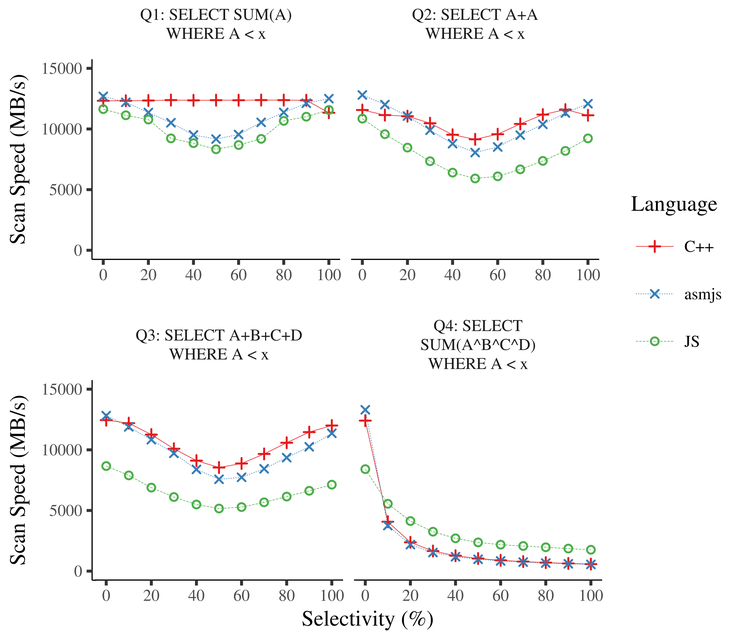
Overhead: JS enter+exit **196 ns** JS to Native call+ret **31 ns**




```
var chase = function (key) { return get(get(key)); }
```

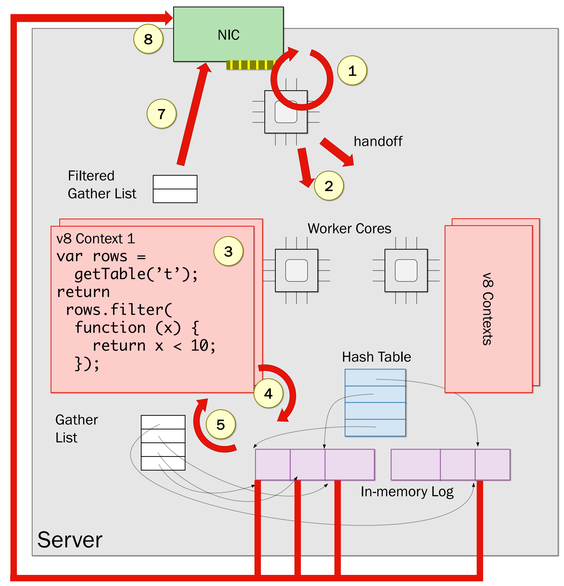
Mechanism	Context Creation	Code Compilation	Context Switch
V8::Context/JS	889 µs	427 µs	98 ns
Processes/C++	763 µs	58,000 µs	1,121 ns
VMFUNC/C++		58,000 µs	138 ns
sthreads/C++	2 µs	58,000 µs	150 ns

Performance Overheads



- Compared to native with no isolation
- JS 18-39% slower than native
- asm.js 2-10% slower than native

Procedure Dispatch



- Not designed for fast dispatch
- Single context can only admit one thread
- Cannot easily move contexts between cores
- Need smart management of context pool

Research Questions

- Right model for extending fast KVS?
- Other approaches
  - Rust type-safety
  - Native Client/SFI
- Dispatch
  - Contexts ↔ Cores
- Garbage collection
  - Regions?
  - Need to roll back globals too?
- Distributed Hotspot information
- JIT-performance vs Optimized C++