Providing Multi-tenant Services with FPGAs: Case Study on a Key-Value Store

Zsolt István*, Gustavo Alonso, Ankit Singla
Systems Group, Computer Science Dept., ETH Zürich
* Now at IMDEA Software Institute, Madrid
FPGAs in the Cloud

• Wider adoption of FPGAs (e.g., Amazon F1, Microsoft Catapult, ...)
• Many promising use-cases but often single-tenant designs

• Clouds built on sharing and multi-tenancy
  ❑ High utilization
  ❑ Flexible provisioning
  ❑ Load isolation and QoS guarantees
Providing multi-tenancy with FPGAs

Virtualization
- General purpose (PR)
- Few tenants
- Trades off functionality
- Course grained resource alloc.
- Tenants “bring” applications

Multi-tenant applications
- Domain-specific
- Many tenants
- Trades off performance (?)
- Fine grained resource alloc.
- Provider “brings” application
Multi-tenant application as a service

Key-value store

• Widely deployed in the cloud and datacenters
• Different tradeoffs but similar interfaces, e.g.:
  • Memcached – caching, no replication, latency-optimized, main-memory
  • Amazon S3 – BLOB store, replicated, BW-optimized, needs large capacity

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>AAA,BBB,CCC</td>
</tr>
<tr>
<td>K2</td>
<td>AAA,BBB</td>
</tr>
<tr>
<td>K3</td>
<td>AAA,DDD</td>
</tr>
<tr>
<td>K4</td>
<td>AAA,2,01/01/2015</td>
</tr>
<tr>
<td>K5</td>
<td>3,ZZZ,5623</td>
</tr>
</tbody>
</table>
Building a multi-tenant KVS (Multes)

• Area well studied in related work
  • Several pipelined designs, all saturate network link
  • **Caribou**: Interfaces and functionality similar to SW [VLDB17]
  • FPGA can provide replication for fault-tolerance [NSDI16]

• Requirements for multi-tenancy:
  • Performance isolation
  • Data isolation
  • Flexibility in resource allocation (focus on network bandwidth)
  • Efficient use of resources regardless of number of tenants

Designing for multi-tenancy

• Caribou is composed of four modules
  • Requests can take various routes
  • Some traffic is inter-node
  • Hard to reason about load interactions!

• Multes: Reorganized pipeline to ensure all requests take same path (1)
  • Hash table implements parts of the replication log features (multi-version)
  • More coupling between modules (op-codes)
Token buckets

- Commonly used in networking scenarios
  - Max. number of tokens ($D$), adding $C$ tokens every $T$ cycles
  - Limits data rate, burst size

- Buffer space on the FPGA?
  - Queue commands before data movement

- Token buckets can be configured with no overhead at runtime (2)
  - Per-tenant allocations controlled by software
Replicated KVS

• Caribou implements inter-FPGA replication (leader based algorithm)
Having multiple roles

• Control state machine at heart of replication protocol
  • Data and control handled separately

• Multiple copies not an option
  • Complex logic + plumbing

• SM extended to store state for each tenant – can context switch per each packet (3)
  • Not all states need tenant context
  • Latency inside SM not on critical path
  • Now in registers, but could use BRAMs to store state
Evaluation of Multes

• Multiple Xilinx VC709s connected to a 10Gbps switch
• 9 load generating machines, Go-based benchmarking tool
  • Tenants connect to different TCP port numbers (e.g. 2880, 2881, ...)

✓ Multes offers flexible multi-tenancy while efficiently using the network link
No performance loss due to multi-tenancy

• Read-only throughput on a single node
Load isolation

• Replicated write latency of Tenant_0 (group = 3)
  • Additional tenants using their full read bandwidth (1/8 of 10Gbps)
The FPGA part on the VC709 is XC7VX690T-2FFG1761C.
Thoughts on the future

Platform-as-a-service

- Customize KVS with tenant-defined processing for different “flavors”

- Combining multi-tenant application with small PR regions
  - Simple streaming interfaces – can use HLS, OpenCL, etc.
  - Misbehaving PR region does not impact others
Conclusion

Multes: multi-tenant KVS service that doesn’t sacrifice performance

Project on Github: https://github.com/fpgasystems/caribou

Relied on three techniques:
1) Single-pipeline architecture and traffic shapers → no load interaction
2) Runtime-parameterization of control modules → flexible allocations
3) “Contexts” in controlling state machines → no overhead when switching between tenants

→ Applicable to many network-facing applications on FPGAs