Cloud Programming Simplified: A Berkeley View on Serverless Computing

Varan Shukla
Background

• Obstacles and Research Opportunities

• Advantages:
  ○ Infinite Compute Resources
  ○ Eliminate up-front commitment
  ○ Pay for use
  ○ Economy of scale
  ○ Simplify Operations
  ○ Increase utilization by multiplexing

→ Proliferation of virtual resources to manage
Introduction

● 2 competing approaches: EC2 vs App Engine
● Why EC2 won?
  ○ Simple to port
● Customers still need to manage the virtual resources themselves.
  ○ Is there an easier path?
● AWS Lambda: cloud functions and serverless (an oxymoron)
● BaaS: Backend as a Service
  ○ specialized frameworks for app specific requirements
    ◆ Serverless = Faas + Baas
    ◆ Autoscale, billed on actual usage
## Serverful vs Serverless

### Parallels Low-level assembly vs High-level Programming

<table>
<thead>
<tr>
<th>Characteristic</th>
<th><strong>AWS Serverless Cloud</strong></th>
<th><strong>AWS Serverful Cloud</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROGRAMMER</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>When the program is run</td>
<td>On event selected by Cloud user</td>
<td>Continuously until explicitly stopped</td>
</tr>
<tr>
<td>Programming Language</td>
<td>JavaScript, Python, Java, Go, C#, etc.</td>
<td>Any</td>
</tr>
<tr>
<td>Program State</td>
<td>Kept in storage (stateless)</td>
<td>Anywhere (stateful or stateless)</td>
</tr>
<tr>
<td>Maximum Memory Size</td>
<td>0.125 - 3 GiB (Cloud user selects)</td>
<td>0.5 - 1952 GiB (Cloud user selects)</td>
</tr>
<tr>
<td>Maximum Local Storage</td>
<td>0.5 GiB</td>
<td>0 - 3600 GiB (Cloud user selects)</td>
</tr>
<tr>
<td>Maximum Run Time</td>
<td>900 seconds</td>
<td>None</td>
</tr>
<tr>
<td>Minimum Accounting Unit</td>
<td>0.1 seconds</td>
<td>60 seconds</td>
</tr>
<tr>
<td>Price per Accounting Unit</td>
<td>$0.00000002 (assuming 0.125 GiB)</td>
<td>$0.0000867 - $0.4080000</td>
</tr>
<tr>
<td>Operating System &amp; Libraries</td>
<td>Cloud provider selects^5</td>
<td>Cloud user selects</td>
</tr>
<tr>
<td><strong>SYSADMIN</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server Instance</td>
<td>Cloud provider selects</td>
<td>Cloud user selects</td>
</tr>
<tr>
<td>Scaling^6</td>
<td>Cloud provider responsible</td>
<td>Cloud user responsible</td>
</tr>
<tr>
<td>Deployment</td>
<td>Cloud provider responsible</td>
<td>Cloud user responsible</td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>Cloud provider responsible</td>
<td>Cloud user responsible</td>
</tr>
<tr>
<td>Monitoring</td>
<td>Cloud provider responsible</td>
<td>Cloud user responsible</td>
</tr>
<tr>
<td>Logging</td>
<td>Cloud provider responsible</td>
<td>Cloud user responsible</td>
</tr>
</tbody>
</table>
Serverful vs Serverless

- Applications
  - Web APIs
  - Event Data Processing
  - Future Serverless Applications

- Serverless
  - Cloud Functions
  - Object Storage
  - Key-Value Database
  - Mobile Backend Database
  - Big Data Query
  - Big Data Transform
  - Messaging
  - Future Serverless Cloud Services

- Base Cloud Platform
  - VM
  - VPC
  - Block Storage
  - IAM
  - Billing
  - Monitoring

- Hardware
  - Server
  - Network
  - Storage
  - Accelerator
Key Distinctions

- Decoupled Computation and Storage
  - Stateless Computation and independent scaling
- Execute without managing resource allocation
  - Auto provisioning
- Pay in proportion to actual usage of resource rather than for allocation

Merely a re-branding of previous offerings like PaaS? Nope
  - Better autoscaling, strong isolation, platform flexibility & ecosystem support
Autoscaling How?

- Need Strong performance & security isolation
- Warm pool of VM instances
- Leverage Unikernels, library OSes, language based VMs, microVMs: Firecracker

How this relates to Kubernetes?

K8S lies somewhere in between - perfect match to hybrid solutions

Serverless is a paradigm shift - fully offloading operational responsibilities
Why is serverless attractive?

- Draw in new customers - makes cloud approachable & easier
- Utilize unused resources
- Increased programming productivity
- Opportunities for software/hardware optimizations & research
- Fine grained accounting (~100ms)
<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
<th>Challenges</th>
<th>Workarounds</th>
<th>Cost-performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-time video compression (ExCamera)</td>
<td>On-the-fly video encoding</td>
<td>Object store too slow to support fine-grained communication; functions too coarse grained for tasks.</td>
<td>Function-to-function communication to avoid object store; a function executes more than one task.</td>
<td>60x faster, 6x cheaper versus VM instances.</td>
</tr>
<tr>
<td>MapReduce</td>
<td>Big data processing (Sort 100TB)</td>
<td>Shuffle doesn’t scale due to object stores latency and IOPS limits</td>
<td>Small storage with low-latency, high IOPS to speed-up shuffle.</td>
<td>Sorted 100 TB 1% faster than VM instances, costs 15% more.</td>
</tr>
<tr>
<td>Linear algebra (Numpy-wren)</td>
<td>Large scale linear algebra</td>
<td>Need large problem size to overcome storage (S3) latency, hard to implement efficient broadcast.</td>
<td>Storage with low-latency high-throughput to handle smaller problem sizes.</td>
<td>Up to 3x slower completion time. 1.26x to 2.5x lower in CPU resource consumption.</td>
</tr>
<tr>
<td>ML pipelines (Cirrus)</td>
<td>ML training at scale</td>
<td>Lack of fast storage to implement parameter server; hard to implement efficient broadcast, aggregation.</td>
<td>Storage with low-latency, high IOPS to implement parameter server.</td>
<td>3x-5x faster than VM instances, up to 7x higher total cost.</td>
</tr>
<tr>
<td>Databases (Serverless SQLite)</td>
<td>Primary state for applications (OLTP)</td>
<td>Lack of shared memory, object store has high latency, lack of support for inbound connectivity.</td>
<td>Shared file system can work if write needs are low.</td>
<td>3x higher cost per transaction than published TPC-C benchmarks. Reads scale to match but writes do not.</td>
</tr>
</tbody>
</table>
MapReduce: their thoughts

- Shuffle operation is a challenge with $M \times R$ transfers
- 100TB of data, 3GB blocks, 33k blocks, 2.22 billion IOPS
- $12,000 in S3 alone
- Solution: Use High performance but much expensive storage (ElastiCache)
- Divide in stages to reduce storage size
- 2983s for $144 on 395 VMs v/s 2945s for $163 using AWS Lambda
Limitations

- Inadequate storage for fine-grained operations
  - calls for development of ephemeral and durable storage
- Lack of fine-grained coordination
  - Calls for VM-based rendezvous server/notification systems
  - Name function instances & allow direct addressability to access internal state
Limitations

- Poor performance for standard communication patterns
  - Shuffle is worst with \((N \times K)^2\) messages v/s \(N^2\)
  - No control of location
- Predictable performance
  - Cold start latency
  - Variable Hardware Resources
What Serverless Computing Should Become

Note challenges in 5 areas:

● Abstraction Challenges
  ○ Resource requirements
    ■ Explicit control? Against the spirit
    ■ Instead raise the level of abstraction: infer eg. static code analysis
  ○ Data Dependencies
    ■ Suboptimal placement = inefficient communication
    ■ Specify its computation graph
What Serverless Computing Should Become

● System Challenges
  ○ High-performance, affordable, transparently provisioned storage
    ■ Ephemeral Storage - dist. In-memory service leveraging statistical multiplexing
    ■ Durable Storage - transparently provisioned
  ○ Coordination/Signaling Service
  ○ Minimise Startup time
    ■ Unikernels, warm pools, incremental loading

● Computer Architecture Challenges
  ○ Hardware Heterogeneity, Pricing, and Ease of Management
    ■ Hardware-software co-design
    ■ Domain Specific Architectures(GPUs, TPUs)
What Serverless Computing Should Become

- Networking Challenges
  - $K^2$ more messages when $K$ functions on a VM
  - Ways to address
    - Combine and share over a single VM
    - Explicit placement
    - Co-locate with computation graph
  - Arguably against the spirit, rescue flexibility for provider

- Security Challenges
  - Scheduling randomization and physical isolation
    - Co-residency attacks are difficult
  - Fine-grained security contexts
  - Oblivious serverless to prevent leaking access patterns
Fallacies and Pitfalls

- More Expensive: could actually end up costing much less
- Unpredictable Costs: bucket based pricing
- Easy to port: not really, need a standard
- Vendor lock in strong: cross-cloud support
- Cannot handle low-latency apps needing predictable performance: not really
- Few “elastic” services are actually as flexible
Predictions

● Serverless skyrockets while serverful although not disappear, will decline
● Expect new BaaS storage services facilitated by serverful
● Will be simpler and more secure
● Billing models will evolve

● Serverless computing will become the default computing paradigm of the Cloud Era, largely replacing serverful computing and thereby bringing closure to the Client-Server Era.
Thank You

Questions?