No More, No Less
A Formal Model for Serverless Computing

Maurizio Gabbrielli, Ivan Lanese, Stefano Pio Zingaro
INRIA, France / Università di Bologna, Italy

Saverio Giallorenzo, Fabrizio Montesi, Marco Peressotti
University of Southern Denmark, Denmark

Coordination 2019
A gentle introduction to “serverless”...

Adapted from “Serverless Architecture Patterns” by Abby Fuller, AWS
A gentle introduction to “serverless”...

- **Monolith**: provisioned, pay-per-deployment
- **Microservices**: on-demand, pay-per-execution
- **Serverless**: 

![Diagram showing Monolith, Microservices, and Serverless models with corresponding icons and descriptions.](image-url)
A gentle introduction to “serverless”...

Monolith

Microservices

Serverless
A gentle introduction to “serverless”...

Monolith

Microservices

Serverless

Still rough around the edges

?!
A gentle introduction to “serverless”...

Autodesk Goes Serverless in the AWS Cloud, Reduces Account-Creation Time by 99%

https://aws.amazon.com/serverless/
A gentle introduction to “serverless”...

Tailor

an excerpt

https://github.com/alanwill/aws-tailor
... and its current limitations (non-exhaustive)

- Currently ~15' execution timeout;
- No function-to-function invocation. Functions need an in-between stateful service to call each other;
- Sparse coordination logic.
... and its current limitations (non-exhaustive)

- Traffic-dependent scaling of functions implies:
  - complex cost model;
  - complex system load estimations.

- Poor performance for standard communication patterns

---

... and its current limitations *(non-exhaustive)*

Lock-in due to absence of standards on:

- function support/execution environments;
- function call semantics;
- semantics of stateful services.
Direction

- We want to study S.C. avoiding any vendor/technology specifics
- We need a formal model for Serverless Computing that
  - Captures current incarnations of S.C.
    - (e.g. event-based, storage-mediated as in AWS)
  - Supports proposed approaches/features
    - (e.g. function-to-function invocation, updatable function definitions)
Serverless Kernel Calculus (SKC)
Serverless Kernel Calculus (SKC)

- **Systems:** \( \langle S, D \rangle \)
Serverless Kernel Calculus (SKC)

- Systems:

  $\langle S, D \rangle$

- Function definitions:

  $f \mapsto M$

  - Repository of function definitions
  - Function name
  - $\lambda$-term
Serverless Kernel Calculus (SKC)

- **Systems:** \( \langle S, D \rangle \)
  - Network of running functions
  - Repository of function definitions

- **Function definitions:** \( f \mapsto M \)
  - Function name
  - \( \lambda \)-term

- **Running functions:** \( c \triangleleft M \)
  - Promise
  - \( \lambda \)-term
SKC function terms

\[ M ::= M \cdot M' \mid V \]

\[ V ::= x \mid \lambda x. M \]
SKC function terms

\[ M ::= M M' \mid V \]

Function invocation

\[ f \]

Asynchronous eval

\[ \langle \mathcal{E}[f], \mathcal{D}[f \mapsto M] \rangle \rightarrow \langle \mathcal{E}[M], \mathcal{D} \rangle \]

Function Repository

\[ \langle \mathcal{E}[f], \mathcal{D}[f \mapsto M] \rangle \rightarrow \langle \mathcal{E}[M], \mathcal{D} \rangle \]

Updates

\[ \langle \mathcal{E}[f], \mathcal{D}[f \mapsto M] \rangle \rightarrow \langle \mathcal{E}[M], \mathcal{D} \rangle \]

Futures

\[ \langle \mathcal{E}[f], \mathcal{D}[f \mapsto M] \rangle \rightarrow \langle \mathcal{E}[M], \mathcal{D} \rangle \]

\[ V ::= x \mid \lambda x. M \]
SKC function terms

\[ \mathcal{M} ::= \mathcal{M} \mathcal{M}' \mid \mathcal{V} \mid f \mid \text{async } \mathcal{M} \mid \text{set } f \mathcal{M} \mid \text{take } f \]

**Function invocation**

| \mathcal{F} | \mathcal{S}, \mathcal{D} \rightarrow \mathcal{S}[\mathcal{V}/f], \mathcal{D} |

**Asynchronous eval**

| \langle \delta[\text{async } \mathcal{M}], \mathcal{D} \rangle \rightarrow \langle \delta[c] \mid c \downarrow \mathcal{M}, \mathcal{D} \rangle |

**Futures**

| \mathcal{C} | \mathcal{S}, \mathcal{D} \rightarrow \mathcal{S}[\mathcal{V}/c], \mathcal{D} |

\[ \mathcal{V} ::= x \mid \lambda x. \mathcal{M} \]
SKC function terms

\[ M ::= M M' \mid V \]

Function invocation
\[ \mid f \]

Asynchronous eval
\[ \mid \text{async } M \]

Function Repository Updates
\[ \mid \text{set } f M \rightarrow \langle \emptyset \text{[set } f \text{ M]}, D \rangle \rightarrow \langle \emptyset \text{[f]}, D[f \mapsto M] \rangle \]
\[ \mid \text{take } f \rightarrow \langle \emptyset \text{[take } f \text{]}, D[f \mapsto M] \rangle \rightarrow \langle \emptyset \text{[M]}, D \setminus f \rangle \]

\[ V ::= x \mid \lambda x. M \]

Futures
\[ \mid c \]
SKC function terms

\[
M ::= M \, M' \mid V \\
\mid f \quad \langle \mathcal{E}[f], \mathcal{D}[f \mapsto M] \rangle \to \langle \mathcal{E}[M], \mathcal{D} \rangle \\
\mid \text{async } M \quad \langle \mathcal{E}[\text{async } M], \mathcal{D} \rangle \to \langle \mathcal{E}[c] \mid c \triangleleft M, \mathcal{D} \rangle \\
\mid \text{set } f \, M \quad \langle \mathcal{E}[\text{set } f \, M], \mathcal{D} \rangle \to \langle \mathcal{E}[f], \mathcal{D}[f \mapsto M] \rangle \\
\mid \text{take } f \quad \langle \mathcal{E}[\text{take } f], \mathcal{D}[f \mapsto M] \rangle \to \langle \mathcal{E}[M], \mathcal{D} \setminus f \rangle \\
\]

\[
V ::= x \mid \lambda x. M \\
\mid c \quad \langle c \triangleleft V \mid S, \mathcal{D} \rangle \to \langle S[V/c], \mathcal{D} \rangle \\
\]
Programmable events in SKC

- Store handlers for event $e$ in the definition repository $\mathcal{D}$:

  \[
  \text{install}_\text{handler} \rightarrow \lambda e. \lambda \text{handler.} \\
  \quad \text{let } \text{old_handler} = \text{take } e \\
  \quad \text{let } \text{new_handler} = \lambda v. \text{do async (handler } v) \\
  \quad \text{(current_handler } v) \\
  \quad \text{set } e \text{ new_handler}
  \]

- Raise event $e$ (with $v$) by invoking its handlers in $\mathcal{D}$:

  \[ e \; v \]

- (See the paper for Tailor in SKC)
Programmable events in SKC

- Store handlers for event $e$ in the definition repository $\mathcal{D}$:

  ```
  install_handler \mapsto \lambda e. \lambda \text{handler}.
  let old_handler = take e
  let new_handler = \lambda v. do async (handler v) (current_handler v)
  set e new_handler
  ```

- Raise event $e$ (with $v$) by invoking its handlers in $\mathcal{D}$:

  ```
  e \ v
  ```

- (See the paper for Tailor in SKC)
Programmable events in SKC

- Store handlers for event e in the definition repository $\mathcal{D}$:

\[
\text{install\_handler } \mapsto \lambda e. \lambda \text{handler}.
\]

\[
\text{let } \text{old\_handler} = \text{take } e
\]

\[
\text{let } \text{new\_handler} = \lambda v. \text{do } \text{async} (\text{handler } v) (\text{current\_handler } v)
\]

\[
\text{set } e \text{ new\_handler}
\]

- Raise event e (with v) by invoking its handlers in $\mathcal{D}$:

\[
e v
\]

- (See the paper for Tailor in SKC)
Conclusions and future work

We introduced SKC a Kernel Calculus for Serverless Computing:
- Build on established models (λ-calculus + futures + function repository)
- captures current programming models
- supports next-gen features e.g. function-to-function invocation

<table>
<thead>
<tr>
<th>Serverless: current challenges</th>
<th>SKC: research direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>The coordination logic is spare and loosely-consistent</td>
<td>Choreographic Programming targeting SKC.</td>
</tr>
<tr>
<td>Estimation of performance and costs is complex</td>
<td>Quantitative SKC</td>
</tr>
</tbody>
</table>
Thanks for your attention

No More, No Less

\[ \lambda + \text{Futures} + \text{Function Repository} = \]

A formal model for Serverless Computing