The Servers of Serverless Computing

A Formal Revisitation of Functions as Services

Saverio Giallorenzo$^{1,2_{\text{former}}}$, Ivan Lanese$^1$, Fabrizio Montesi$^2$, Davide Sangiorgi$^1$, and Stefano Pio Zingaro$^1$

$^1$Università di Bologna/INRIA
$^2$University of Southern Denmark
A Gentle Introduction to Serverless Computing

Monolith

Microservices

Serverless

Function

Software Unit

Runtime Environment
A Gentle Introduction to Serverless

Optimisation and Security

Execution Scheduling

Deployment

Heterogeneous storage/messaging patterns

Programming model

Programming Layer

Implementation Layer
Serverless as Research Topic

Source: Scopus
## Serverless as Research Topic

<table>
<thead>
<tr>
<th>Venue</th>
<th># Papers</th>
<th>Core / SCIMAGO Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Future Generation Computer Systems</td>
<td>8</td>
<td>Software : Q1</td>
</tr>
<tr>
<td>IEEE Internet Computing</td>
<td>3</td>
<td>Computer Networks and Communications : Q1</td>
</tr>
<tr>
<td>IEEE Transactions on Parallel and Distributed Systems</td>
<td>2</td>
<td>Computational Theory and Mathematics: Q1</td>
</tr>
<tr>
<td>USENIX Annual Technical Conference + HotCloud</td>
<td>13 (6,7)</td>
<td>A / -</td>
</tr>
<tr>
<td>IC2E + IEEE CLOUD + CLOSER</td>
<td>20 (5,10,5)</td>
<td>- / B / -</td>
</tr>
<tr>
<td>ACM Symposium on Cloud Computing (SoCC)</td>
<td>12</td>
<td>-</td>
</tr>
<tr>
<td>SIGMOD</td>
<td>4</td>
<td>A*</td>
</tr>
<tr>
<td>Middleware</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>CIDR</td>
<td>3</td>
<td>A</td>
</tr>
<tr>
<td>OOPSLA</td>
<td>2</td>
<td>A*</td>
</tr>
<tr>
<td>ICSE</td>
<td>2</td>
<td>A*</td>
</tr>
<tr>
<td>INFOCOM</td>
<td>2</td>
<td>A*</td>
</tr>
</tbody>
</table>

Source: DBLP
The Servers of Serverless Computing

Programming Layer

Implementation Layer

SKC
influenced by \(\lambda\) and \(\pi\) calculus

\(\pi\) calculus
Our new version of SKC (Section 2) provides a few improvements, which are given by a better integration of the essential features of the $\lambda$-calculus and the $\mu$-calculus. Specifically, functions can now be parametric on the names of other functions available in a serverless system, whereas before all references to functions in the repository were statically fixed; it is now possible to create new function names for the repository dynamically, so the repository of available functions can now grow freely at runtime.

These new features enhance the expressiveness of the language, which we illustrate through small examples (Section 2) and two use cases (Section 3) from artificial intelligence, one implementing the perceptron algorithm and one for distributed tagging of large images.

We present two semantic interpretations for our version of SKC. The first (Section 2) is a refinement of the original reduction semantics from [13], which supports the aforementioned improvements. This high-level semantics is intended for developers to reason abstractly about SKC programs. The second semantic interpretation is a formalisation of a possible implementation layer for SKC, given in terms of an encoding (Section 5) from SKC to the asynchronous $\mu$-calculus [41] (recalled in Section 4). The encoding is inspired by Milner's encoding of the call-by-value $\lambda$-calculus [26]. It shows how serverless functions can be implemented by servers (replicated processes in the $\mu$-calculus) that can be triggered by messages from clients, and how a serverless implementation layer can be modelled in terms of communications among processes. We prove that the encoding is correct in terms of an operational correspondence result.

Our results show that standard techniques from process calculi can be useful to understand the two layers of serverless calculi. Hopefully, this understanding could also provide foundations for tackling some outstanding questions in serverless computing. For example, predicting resource usage and costs is challenging in general, since it requires knowing how functions are executed by the implementation layer.

### Figure 1

Syntax of SKC.

<table>
<thead>
<tr>
<th>Configurations</th>
<th>$C ::= \langle S, D \rangle \mid \nu n C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition repository</td>
<td>$D ::= {(f_1, M_1), \ldots, (f_k, M_k)}$ (k $\geq$ 0)</td>
</tr>
<tr>
<td>Systems</td>
<td>$S, S' ::= c \blacktriangle M \mid S \mid S' \mid \nu n S \mid 0$</td>
</tr>
<tr>
<td>Functions</td>
<td>$M, N ::= M \mid N \mid V \mid$</td>
</tr>
<tr>
<td></td>
<td>$\quad \quad \quad \quad \quad \quad \quad \text{call } h \mid \text{store } h \mid N \mid M \mid \text{take } h \mid \nu f M \mid \text{async } M \mid c$</td>
</tr>
<tr>
<td>Values</td>
<td>$V, V' ::= x \mid \lambda x. M \mid f$</td>
</tr>
<tr>
<td>Restrictable names</td>
<td>$n ::= c \mid f$</td>
</tr>
<tr>
<td></td>
<td>$h ::= f \mid x$</td>
</tr>
<tr>
<td>Function names</td>
<td>$f \in \text{Fun}$</td>
</tr>
<tr>
<td>Future names</td>
<td>$c \in \text{Fut}$</td>
</tr>
<tr>
<td>Variables</td>
<td>$x \in \text{Var}$</td>
</tr>
</tbody>
</table>
SKC • Simple Example

\[
\langle c \downarrow \text{call } f, \ D \cup \{ (f, M) \} \rangle
\]

\[
\rightarrow \langle c \downarrow M, \ D \rangle
\]

\[
\rightarrow \langle c \downarrow V, \ D \rangle
\]
SKC • Simple Example (async)

\[ \langle c \triangleright \text{async call } f, D \rangle \]
\[ \rightarrow \langle \nu c' (c \triangleright c' \mid c' \triangleright \text{call } f), D \rangle \]
\[ \rightarrow \langle \nu c' (c \triangleright c' \mid c' \triangleright V), D \rangle \]
\[ \rightarrow \langle \nu c' (c \triangleright V \mid c' \triangleright V), D \rangle \]
SKC • Example, Private State

\[
(\text{newLog}, \nu\text{log}(\text{store log call nil log})) \in D
\]

- Fresh name/restriction
- Name
- Body
- Continuation

Empty list
SKC • Example, Private State

\( \text{newLog, vlog(store log call nil log)} \) \( \in D \)
SKC · Example, Private State

\[
\left( \text{newLog}, \nu \text{log}(\text{store log call \(\text{nil log}\)))} \right) \in D
\]

\[
\langle c \succeq (\lambda x. (\text{call pair ((M x)(N x)) x})) \text{ call newLog, } D \rangle
\]
SKC • Example, Private State

( newLog, vlog(store log call nil log) ) ∈ D

\[ \langle c \downarrow (\lambda x. (\text{call pair } ((M \; x)(N \; x)) \; x)) \rangle\quad \text{call newLog, } D \]

\[ \langle c \downarrow (\lambda x. (\text{call pair } ((M \; x)(N \; x)) \; x))\rangle\quad \text{vlog(store log call nil log), } D \]
SKC • Example, Private State

\[
(\text{newLog}, \text{vlog}\text{(store log call nil log)}) \in D
\]

\[
\langle c\downarrow(\lambda x. (\text{call pair ((M x)(N x)) x})) \rangle \quad \text{call newLog, } D
\]

\[
\langle c\downarrow(\lambda x. (\text{call pair ((M x)(N x)) x})) \rangle \quad \text{vlog}\text{(store log call nil log), } D
\]

\[
\text{vlog}\langle c\downarrow(\lambda x. (\text{call pair ((M x)(N x)) x})) \rangle \quad \text{log, } D \cup \{(\text{log, call nil})\}
\]
SKC · Example, Private State

\((\text{newLog}, \nu\text{log}(\text{store log call nil log})) \in D)\)

\[\langle c \downarrow (\lambda x. (\text{call pair } ((M x)(N x)) x)) \rangle \text{ call newLog, } D \rangle\]

\[\langle c \downarrow (\lambda x. (\text{call pair } ((M x)(N x)) x)) \rangle \nu\text{log}(\text{store log call nil log}), D \rangle\]

\[\nu\text{log}\{c \downarrow (\lambda x. (\text{call pair } ((M x)(N x)) x)) \rangle \text{ log, } D \cup \{(\text{log, call nil})\}\}\]

\[\nu\text{log}\{c \downarrow \text{call pair } ((M \text{log})(N \text{log})) \text{ log, } D \cup \{(\text{log, call nil})\}\}\]
SKC • Example, Private State

\[
(\text{newLog}, \nu\text{log}(\text{store } \log \text{ call nil } \log)) \in D
\]

\[
\langle c \langle \lambda x. (\text{call } \text{pair } ((M x)(N x)) x) \rangle \rangle \text{ call newLog, } D \rangle
\]

\[
\langle c \langle \lambda x. (\text{call } \text{pair } ((M x)(N x)) x) \rangle \rangle \nu\text{log}(\text{store } \log \text{ call nil } \log), D \rangle
\]

\[
\nu\text{log}\langle c \langle \lambda x. (\text{call } \text{pair } ((M x)(N x)) x) \rangle \rangle \text{ log}, D \cup \{(\log, \text{call nil})\}
\]

\[
\nu\text{log}\langle c \langle \text{call } \text{pair } ((M \log)(N \log)) \log \rangle \rangle \text{ log}, D \cup \{(\log, \text{call nil})\}
\]

\[
\nu\text{log}\langle c \langle \text{call } \text{pair } (M \log V_N \log) \rangle \log, D \cup \{(\log, N_{\log})\}\rangle
\]
SKC • Example, Private State

\((\text{newLog}, \nu\log(\text{store log call nil log})) \in D\)

\[\langle c \cdot (\lambda x. (\text{call pair } ((M x)(N x)) x)) \rangle \text{ call newLog, } D\]

\[\langle c \cdot (\lambda x. (\text{call pair } ((M x)(N x)) x)) \rangle \nu\log(\text{store log call nil log}), D\]

\[\nu\log\langle c \cdot (\lambda x. (\text{call pair } ((M x)(N x)) x)) \rangle \text{ log, } D \cup \{(\text{log, call nil})\}\]

\[\nu\log\langle c \cdot \text{call pair } ((M \log)(N \log)) \rangle \text{ log, } D \cup \{(\text{log, call nil})\}\]

\[\nu\log\langle c \cdot \text{call pair } (M \log V_N) \rangle \text{ log, } D \cup \{(\text{log, } N_{\log})\}\]

\[\nu\log\langle c \cdot \text{call pair } V_M \rangle \text{ log, } D \cup \{(\text{log, } N_{\log} :: M_{\log})\}\]
The Servers of Serverless Computing

SKC • Results, $SKC \leftrightarrow \pi$ Operational Correspondence

Theorem 1. From $SKC$-to-$\pi$ operational correspondence

If $C \rightarrow C'$ then $[[C]]^* \rightarrow \approx [[C']]^*$

Theorem 2. From $\pi$-to-$SKC$ operational correspondence.

If $\{[[C]]\}^* \rightarrow P$ then there is $C'$ with $C \rightarrow C'$ and $P \approx [[C']]^*$
SKC • Future Work

- **guarantees** like sequential execution, sequential consistency, and global-state transformation serialisability;

- **programming models** that give programmers a global view of the overall logic of the distributed functions and capture the loosely-consistent execution model of Serverless;

- **transformation frameworks**, e.g., depending on the application context and inbound load, users/optimisation systems can transform parts of a given system from Serverless to Microservices and vice versa;

- **prediction models** for cost/resource usage, which require a modelling that relates functions and their execution at the implementation layer.
Thank for your time

Happy cruising!
Appendix
SKC · Example, applications and non-determinism

\[
\begin{align*}
\langle c_0 \triangleright \text{store } w & a \ (\text{call } \text{pair} \ (\text{call } \text{cons} \ 0 \ \text{call } \text{cons} \ 0 \ \text{call } \text{nil}) \ 1) \ () \\
| c_1 \triangleright \text{call } \text{trainAndStore} \ w & a \ (\text{call } \text{pair} \ (\text{call } \text{cons} \ 0 \ \text{call } \text{cons} \ 0 \ \text{call } \text{nil}) \ 0) \\
| c_2 \triangleright \text{call } \text{trainAndStore} \ w & a \ (\text{call } \text{pair} \ (\text{call } \text{cons} \ 0 \ \text{call } \text{cons} \ 1 \ \text{call } \text{nil}) \ 0) \\
| c_3 \triangleright \text{call } \text{trainAndStore} \ w & a \ (\text{call } \text{pair} \ (\text{call } \text{cons} \ 1 \ \text{call } \text{cons} \ 0 \ \text{call } \text{nil}) \ 0) \\
| c_4 \triangleright \text{call } \text{trainAndStore} \ w & a \ (\text{call } \text{pair} \ (\text{call } \text{cons} \ 1 \ \text{call } \text{cons} \ 1 \ \text{call } \text{nil}) \ 1) \\
| c_5 \triangleright \lambda w. \ (\text{call } \text{predict} \ (\text{call } \text{cons} \ 0 \ \text{call } \text{cons} \ 1 \ \text{call } \text{nil}) \ (\text{call } \text{first} \ w) \\
\hspace{1cm} (\text{call } \text{second} \ w)) \ \text{call } w, D \rangle
\end{align*}
\]
SKC • Example, applications and non-determinism

\[ \langle c_0 \text{store wa } \cdots \right. \\
| \left. c_1 \text{call } trainAndStore \text{ call wa } \cdots \right. \\
| \left. c_2 \text{call } trainAndStore \text{ call wa } \cdots \right. \\
| \left. c_3 \text{call } trainAndStore \text{ call wa } \cdots \right. \\
| \left. c_4 \text{call } trainAndStore \text{ call wa } \cdots \right. \\
| \left. c_5 \lambda w . (\text{call predict } \cdots ) \text{ call wa, } D \right. \]