Autoscaling Rethought: A Continuous, Decentralized Approach

Martin Straesser
Intelligent Serverless and Cloud Applications Symposium
June 17-18, 2024, Zurich, Switzerland
• Dynamically provision computing resources under varying load

• High-valued task in production systems

• Affects both operating costs and customer experience

• Across several sub-domains: traditional cloud computing, fog computing or serverless computing
Continuous Decentralized Autoscaling - Context

- Vision paper: doi.org/10.1109/CCGridW59191.2023.00058
- Full paper under submission
Continuous Decentralized Autoscaling – The Analogy

Conventional autoscaling
Continuous Decentralized Autoscaling – The Analogy

Conventional autoscaling
Continuous Decentralized Autoscaling – The Analogy

Conventional autoscaling

Never again!

…works, but what if:

There is a sudden increase of customers at an unfortunate time?
Continuous Decentralized Autoscaling – The Analogy

Conventional autoscaling

…works, but what if:
The camera fails?
Continuous Decentralized Autoscaling – The Analogy

Conventional autoscaling

…works, but what if:
The manager fails?
Continuous Decentralized Autoscaling – The Analogy

Conventional autoscaling

Arriving requests / Workload

Monitoring system

Microservices / Functions

Autoscaler
Continuous Decentralized Autoscaling – The Analogy

Decentralized autoscaling

Help me!
Continuous Decentralized Autoscaling – The Analogy

Decentralized autoscaling
Continuous Decentralized Autoscaling – The Analogy

Decentralized autoscaling

All good!

All good!
Continuous Decentralized Autoscaling – The Analogy

Decentralized autoscaling

Bored, I will leave!
Continuous Decentralized Autoscaling – The Analogy

Decentralized autoscaling

Problematic situation

Bored, I will leave!
Continuous Decentralized Autoscaling – The Analogy

Continuous Decentralized Autoscaling
Continuous Decentralized Autoscaling – The Analogy

For a “large” number of instances we converge into a continuous process.

For the example here with 6 cashiers taking decisions every 1-3 mins, the average time to the next decision is 21 seconds when we look at the system at an arbitrary time.
Continuous Decentralized Autoscaling – Key Characteristics

<table>
<thead>
<tr>
<th>Conventional Autoscalers</th>
<th>Continuous Decentralized Autoscaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>One autoscaler manages a group of service instances.</td>
<td>Each instance gets the right to trigger autoscaling.</td>
</tr>
<tr>
<td>Requires centralized monitoring system and data gathering.</td>
<td>Each instance processes own monitoring data, no centralized monitoring needed.</td>
</tr>
<tr>
<td>Monitoring and data processing overhead increases with system size.</td>
<td>This overhead is constant.</td>
</tr>
<tr>
<td>Scaling policies might be fitted to a specific system size.</td>
<td>As instances are not aware of total system size, scaling policies are always scale-invariant.</td>
</tr>
<tr>
<td>Autoscalers must choose scaling action from a large action space.</td>
<td>Instances just decide between UP, DOWN and HOLD.</td>
</tr>
<tr>
<td>Reaction times to unforeseen events are limited by design.</td>
<td>Reaction times to unforeseen events depend on system size.</td>
</tr>
<tr>
<td>Complex algorithms/learning are used to take correct decisions. Wrong decisions are costly.</td>
<td>Wrong scaling decisions are not costly because of the increased scaling frequency.</td>
</tr>
</tbody>
</table>
Continuous Decentralized Autoscaling – Algorithm & Config

Algorithm 1 Algorithm for Decentralized Autoscaling

Input: \( D : M \to [0; 1], U : M \to [0; 1], \) Distribution \( W \)

1. while instance is running do
2. Collect recent monitoring data into \( m_t \)
3. Draw sample \( r \) from a uniform distribution in \([0; 1]\)
4. \( P(UP) = U(m_t) \)
5. if \( P(UP) > 0 \) and \( r < P(UP) \) then
6. \hspace{1em} decision = UP
7. else
8. \hspace{1em} \( P(DOWN) = D(m_t) \)
9. \hspace{1em} if \( P(DOWN) > 0 \) and \( r < P(DOWN) \) then decision = DOWN
10. \hspace{1em} else decision = HOLD
11. end if
12. Submit decision to scaling executor
13. Draw sample \( w \) from \( W \) and wait for time \( w \)
14. end while

- Parameters:
  - \( U \) – Upscaling function
  - \( D \) – Downscaling function
  - \( W \) – Waiting time distribution

- \( U \) and \( D \) take the current instance monitoring data as inputs and output a probability for the corresponding action
In the following, we consider autoscaling based on CPU utilization.

Example policy:

- **P(DOWN)**
  - Tolerance area: 0.3 \( \leq U \leq 0.8
  - D
- **P(UP)**
  - U
  - Tolerance area: 0.3 \( \leq U \leq 0.8
  - U

CPU Utilization

P(DOWN)

P(UP)
Continuous Decentralized Autoscaling – Scaling Policies

Fig. 7. Examples illustrating the influence of various configuration parameters.
Continuous Decentralized Autoscaling - Implementation

Fig. 9. Proof of Concept Implementation.
Continuous Decentralized Autoscaling – Evaluation Highlights

Test: SHA-256 hash function in Google Cloud

Fig. 1. Motivating example of deployed instances over time and respective QoS and cost metrics.
Continuous Decentralized Autoscaling – Evaluation Highlights

➢ There is more in the paper…
   • A model for worst-case assessments using discrete-time analysis
   • A simulation that can be used for tuning configuration parameters
   • More real-world evaluation results, incl. Azure trace workloads in a Knative environment

Fig. 4. Metrics for step response.

<table>
<thead>
<tr>
<th>Scale Factor s</th>
<th>$T_R [s]$</th>
<th>$T_S [s]$</th>
<th>$N_{max}/s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>241.6 ± 24.8</td>
<td>268.3 ± 68.2</td>
<td>1.00 ± 1.26</td>
</tr>
<tr>
<td>5</td>
<td>246.5 ± 9.9</td>
<td>322.9 ± 26.8</td>
<td>0.64 ± 0.23</td>
</tr>
<tr>
<td>10</td>
<td>238.9 ± 10.1</td>
<td>300.0 ± 61.7</td>
<td>0.36 ± 0.42</td>
</tr>
<tr>
<td>50</td>
<td>237.8 ± 7.5</td>
<td>311.4 ± 10.2</td>
<td>0.27 ± 0.03</td>
</tr>
<tr>
<td>100</td>
<td>240.1 ± 6.1</td>
<td>310.2 ± 7.9</td>
<td>0.21 ± 0.05</td>
</tr>
</tbody>
</table>

Peak load: 10,000+ req/s
Thank you!