Karma:
Resource Allocation for Dynamic Demands

Midhul Vuppalapati
Giannis Fikioris
Rachit Agarwal
Asaf Cidon
Anurag Khandelwal
Éva Tardos
Resource allocation is a fundamental problem

Allocating a resource (e.g. CPU, Memory) with a fixed capacity across multiple users
Resource allocation is a fundamental problem
Allocating a resource (e.g. CPU, Memory) with a fixed capacity across multiple users

Risk Desirable Properties
Resource allocation is a fundamental problem
Allocating a resource (e.g. CPU, Memory) with a fixed capacity across multiple users

Resource Pool

Key Desirable Properties

Pareto efficiency
Resources should not remain unused if there is demand
Resource allocation is a fundamental problem

Allocating a resource (e.g. CPU, Memory) with a fixed capacity across multiple users

Resource Pool

Key Desirable Properties

- **Pareto efficiency**: Resources should not remain unused if there is demand.
- **Strategy-proofness**: Selfish users cannot increase their allocation by lying. (selfish ≠ adversarial)
Resource allocation is a fundamental problem

Allocating a resource (e.g. CPU, Memory) with a fixed capacity across multiple users

Key Desirable Properties

Pareto efficiency
Resources should not remain unused if there is demand

Strategy-proofness
Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)

Fairness
Balanced resource allocations across users
Resource allocation is a fundamental problem

Allocating a resource (e.g. CPU, Memory) with a fixed capacity across multiple users

Key Desirable Properties

- **Pareto efficiency**
  - Resources should not remain unused if there is demand

- **Strategy-proofness**
  - Selfish users cannot increase their allocation by lying (selfish ≠ adversarial)

- **Fairness**
  - Balanced resource allocations across users

Two popular resource allocation mechanisms (for single resource type):
  - strict partitioning and max-min fairness
Strict partitioning
Allocating the resource equally across all users ("fair share") \textit{independent of their demands}

Resource Pool
Strict partitioning

Allocating the resource equally across all users ("fair share") independent of their demands
Strict partitioning

Allocating the resource equally across all users ("fair share") independent of their demands
Strict partitioning

Allocating the resource equally across all users ("fair share") independent of their demands
Strict partitioning

Allocating the resource equally across all users (“fair share”) independent of their demands.
Strict partitioning

Allocating the resource equally across all users ("fair share") independent of their demands

Demands  Resource Pool

- Unsatisfied demand

- Wasted resource
Strict partitioning

Allocating the resource equally across all users (“fair share”) independent of their demands

Demands       Resource Pool

- Unsatisfied demand
- Wasted resource

Pareto efficiency: X
Strategy-proofness: ✓
Fairness: ✓
Strict partitioning

Allocating the resource equally across all users (“fair share”) independent of their demands

Demand vs. Resource Pool

- Unsatisfied demand
- Wasted resource

Resource Pool

- Pareto efficiency: ✗
- Strategy-proofness: ✓
- Fairness: ✓

Strict partitioning does not guarantee Pareto efficiency
Max-min Fairness alleviates the limitations of strict partitioning (under an assumption)

Maximizing minimum allocation across users while ensuring allocation $\leq$ demand for each user

Demands    Resource Pool

\begin{figure}
\end{figure}
Max-min Fairness alleviates the limitations of strict partitioning (under an assumption)

Maximizing minimum allocation across users while ensuring allocation ≤ demand for each user
Max-min Fairness alleviates the limitations of strict partitioning (under an assumption)

Maximizing minimum allocation across users while ensuring allocation ≤ demand for each user

Demands | Resource Pool

- Blue
- Pink
- Yellow
Max-min Fairness alleviates the limitations of strict partitioning (under an assumption)

Maximizing minimum allocation across users while ensuring allocation $\leq$ demand for each user

Demands | Resource Pool
---|---

Pareto efficiency

Strategy-proofness

Fairness

Classical result: Max-min fairness satisfies Pareto efficiency, strategy-proofness and fairness

Underlying assumption: User demands are static
Dynamic demands are the norm in real world deployments
Dynamic demands are the norm in real world deployments

Shared Analytics Clusters
Dynamic demands are the norm in real world deployments

Shared Analytics Clusters

In-memory Key-Value Caches
Dynamic demands are the norm in real world deployments

- Spark
- Hadoop
- Redis
- Memcached

Shared Analytics Clusters

In-memory Key-Value Caches

Inter-datacenter Network Links
Dynamic demands are the norm in real world deployments

- Shared Analytics Clusters
- In-memory Key-Value Caches
- Inter-datacenter Network Links

Analysis of real world workloads
Dynamic demands are the norm in real world deployments

Shared Analytics Clusters

In-memory Key-Value Caches

Inter-datacenter Network Links

Analysis of real world workloads

CPU Demand

Memory Demand
Dynamic demands are the norm in real world deployments

Shared Analytics Clusters

In-memory Key-Value Caches

Inter-datacenter Network Links

Analysis of real world workloads

CPU Demand

Memory Demand

2-3x variation in demands over 10s of seconds
Dynamic demands are the norm in real world deployments

Shared Analytics Clusters

In-memory Key-Value Caches

Inter-datacenter Network Links

2-3x variation in demands over 10s of seconds
40-70% of users have standard deviation of demands > 1/2 mean
Dynamic demands are the norm in real world deployments

- Shared Analytics Clusters
- In-memory Key-Value Caches
- Inter-datacenter Network Links

2-3x variation in demands over 10s of seconds
Dynamic demands are the norm in real world deployments

Shared Analytics Clusters
2-3x variation in demands over 10s of seconds

In-memory Key-Value Caches
More than 5x variation in demands within an hour
[Twitter, Yang et al., OSDI’20]

Inter-datacenter Network Links
35% variation in demands over 5 minute intervals
[Microsoft, Abuzaid et al., NSDI’21]
Dynamic demands are the norm in real world deployments

- **Spark**
  - Shared Analytics Clusters
  - 2-3x variation in demands over 10s of seconds

- **redis**
  - In-memory Key-Value Caches
  - More than 5x variation in demands within an hour
    - [Twitter, Yang et al., OSDI'20]

- **Inter-datacenter Network Links**
  - 35% variation in demands over 5 minute intervals
    - [Microsoft, Abuzaid et al., NSDI'21]

Significant variation in user demands over time in real workloads
Max-min fairness under dynamic demands: *looses one or more of its properties*
Max-min fairness under dynamic demands: loses one or more of its properties

Max-min fairness
based on demands at $t=0$

- Pareto efficiency
- Strategy-proofness
- Fairness
Max-min fairness under dynamic demands: loses one or more of its properties

Max-min fairness based on demands at $t=0$

- Pareto efficiency: False
- Strategy-proofness: False
- Fairness: (Explained in paper)
Max-min fairness under dynamic demands: *looses one or more of its properties*

<table>
<thead>
<tr>
<th>Max-min fairness</th>
<th>Max-min fairness applied periodically</th>
</tr>
</thead>
<tbody>
<tr>
<td>based on demands at $t=0$</td>
<td></td>
</tr>
</tbody>
</table>

- **Pareto efficiency**: ✗
- **Strategy-proofness**: ✗
- **Fairness**: (Explain in paper)
Max-min fairness under dynamic demands: loses one or more of its properties

Max-min fairness based on demands at $t=0$

Pareto efficiency

Strategy-proofness

Fairness

Max-min fairness applied periodically
Max-min fairness under dynamic demands: loses one or more of its properties
Max-min fairness under dynamic demands: loses one or more of its properties

Running Example

Max-min fairness based on demands at $t=0$

Pareto efficiency

Strategy-proofness

Fairness

Max-min fairness applied periodically
Max-min fairness under dynamic demands: **looses one or more of its properties**

Running Example

<table>
<thead>
<tr>
<th>Demands</th>
<th>Resource Pool</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Time

0 | 1 | 2

Max-min fairness based on demands at t=0

Max-min fairness applied periodically

- Pareto efficiency
- Strategy-proofness
- Fairness
Max-min fairness under dynamic demands: loses one or more of its properties

Running Example

<table>
<thead>
<tr>
<th>Demands</th>
<th>Resource Pool</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Max-min fairness based on demands at t=0 applied periodically

- Pareto efficiency
- Strategy-proofness
- Fairness

Max-min fairness under dynamic demands: loses one or more of its properties.
Max-min fairness under dynamic demands: loses one or more of its properties

Running Example

Demands | Resource Pool
---|---
5 | 4
4 | 3
3 | 2
2 | 1
1 |

Time

Max-min fairness based on demands at t=0

Max-min fairness applied periodically

- Pareto efficiency
- Strategy-proofness
- Fairness

Max-min fairness under dynamic demands: loses one or more of its properties
Running Example

Max-min fairness under dynamic demands: looses one or more of its properties

- Max-min fairness based on demands at t=0
- Max-min fairness applied periodically

- Pareto efficiency
- Strategy-proofness
- Fairness

Diagram showing demands and resource pool over time.
Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

Demands

Resource Pool

Max-min fairness based on demands at \( t=0 \)

Max-min fairness applied periodically

Pareto efficiency

Strategy-proofness

Fairness
Max-min fairness under dynamic demands: loses one or more of its properties

Running Example

Max-min fairness based on demands at $t=0$

Max-min fairness applied periodically

Pareto efficiency

Strategy-proofness

Fairness
Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

Max-min fairness based on demands at $t=0$

Max-min fairness applied periodically

Pareto efficiency

Strategy-proofness

Fairness
Max-min fairness under dynamic demands: loses one or more of its properties

Running Example

Max-min fairness based on demands at t=0

Max-min fairness applied periodically

Pareto efficiency

Strategy-proofness

Fairness
Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

Max-min fairness based on demands at $t=0$

Max-min fairness applied periodically

Pareto efficiency
Strategy-proofness
Fairness
Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

Demands Resource Pool

Max-min fairness based on demands at $t=0$

Max-min fairness applied periodically

Pareto efficiency

Strategy-proofness

Fairness
Max-min fairness under dynamic demands: loses one or more of its properties

Running Example

<table>
<thead>
<tr>
<th>Demands</th>
<th>Resource Pool</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Max-min fairness based on demands at $t=0$

Max-min fairness applied periodically

- Pareto efficiency
- Strategy-proofness
- Fairness
Max-min fairness under dynamic demands: loses one or more of its properties

Running Example

<table>
<thead>
<tr>
<th>Time</th>
<th>Demands</th>
<th>Resource Pool</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Max-min fairness based on demands at t=0 applied periodically

- Pareto efficiency: X
- Strategy-proofness: X
- Fairness: X

Max-min fairness applied periodically: loses one or more of its properties
Max-min fairness under dynamic demands: loses one or more of its properties

Running Example

Max-min fairness based on demands at $t=0$

Max-min fairness applied periodically

Pareto efficiency

Strategy-proofness

Fairness
Max-min fairness under dynamic demands: loses one or more of its properties

Running Example

Max-min fairness based on demands at t=0 applied periodically

- Pareto efficiency
- Strategy-proofness
- Fairness

Max-min fairness under dynamic demands: applies periodically

Pareto efficiency ✓
Strategy-proofness ✓
Fairness ✓
Max-min fairness under dynamic demands: loses one or more of its properties

Running Example

Demands | Total Allocation
---|---

Max-min fairness based on demands at $t=0$

Max-min fairness applied periodically

Pareto efficiency

Strategy-proofness

Fairness
Running Example

Max-min fairness under dynamic demands: looses one or more of its properties

- Max-min fairness based on demands at t=0
- Max-min fairness applied periodically

- Pareto efficiency
- Strategy-proofness
- Fairness

2x Gap
Max-min fairness under dynamic demands: loses one or more of its properties

For \( n \) users, a user can get \( \Omega(n) \) more allocation than others

Running Example

Max-min fairness based on demands at \( t=0 \)

Max-min fairness applied periodically

Pareto efficiency

Strategy-proofness

Fairness

2x Gap
Max-min fairness under dynamic demands: loses one or more of its properties

For $n$ users, a user can get $\Omega(n)$ more allocation than others

Running Example

Demands

Total Allocation

For users, a user can get $\Omega(n)$ more allocation than others
Max-min fairness under dynamic demands: loses one or more of its properties

Running Example

Demands | Total Allocation

Max-min fairness based on demands at t=0

Pareto efficiency: x

Strategy-proofness: x

Fairness: x

For n users, a user can get $\Omega(n)$ more allocation than others
Max-min fairness under dynamic demands: loses one or more of its properties

For n users, a user can get $\Omega(n)$ more allocation than others

Max-min fairness based on demands at $t=0$ applied periodically

Pareto efficiency: $\times$

Strategy-proofness: $\times$

Fairness: $\times$

Need to revisit classical resource allocation problem under dynamic demands
Karma: Revisiting the classical resource allocation problem under dynamic demands
Karma: Revisiting the classical resource allocation problem under dynamic demands

New resource allocation algorithm for dynamic demands
Karma: Revisiting the classical resource allocation problem under dynamic demands

New resource allocation algorithm for dynamic demands

Theoretical guarantees

- Pareto efficiency
- Strategy-proofness
- Fairness
Karma: Revisiting the classical resource allocation problem under dynamic demands

New resource allocation algorithm for dynamic demands

Theoretical guarantees

- Pareto efficiency
- Strategy-proofness
- Fairness

Prototype implementation and evaluation
Karma: Revisiting the classical resource allocation problem under dynamic demands

New resource allocation algorithm for dynamic demands

Theoretical guarantees

Pareto efficiency

Strategy-proofness

Fairness

Prototype implementation and evaluation
Karma key idea #1: *donated slices* and *shared slices*
Karma key idea #1: *donated slices* and *shared slices*
Karma key idea #1: *donated slices* and *shared slices*
Karma key idea #1: *donated slices* and *shared slices*
Karma key idea #1: *donated slices* and *shared slices*
Karma key idea #1: *donated slices* and *shared slices*
Karma key idea #1: *donated slices* and *shared slices*
Karma key idea #2: “credits” for resource allocation
Karma key idea #2: “credits” for resource allocation

-1 credit for borrowing a slice
Karma key idea #2: “credits” for resource allocation

-1 credit for **borrowing** a slice

+1 credit if a **donated** slice is borrowed
Karma key idea #2: “credits” for resource allocation

-1 credit for borrowing a slice

+1 credit if a donated slice is borrowed

(Every user is given initial credits at t=0)
Karma key idea #2: “credits” for resource allocation

-1 credit for **borrowing** a slice

+1 credit if a **donated** slice is borrowed

(Every user is given initial credits at t=0)
Karma key idea #2: “credits” for resource allocation

-1 credit for **borrowing** a slice

+1 credit if a **donated** slice is borrowed

(Every user is given initial credits at t=0)
Karma key idea #2: “credits” for resource allocation

-1 credit for **borrowing** a slice

+1 credit if a **donated** slice is borrowed

(Every user is given initial credits at t=0)
Karma allocation algorithm
Karma allocation algorithm

Pick **borrower** with **maximum** credits
Karma allocation algorithm

- Pick **borrower** with **maximum** credits
- Pick **donor** with **minimum** credits
Karma allocation algorithm

- Pick **borrower** with **maximum** credits
- Pick **donor** with **minimum** credits
  - If no donors, then use a shared slice
Karma allocation algorithm

Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits
   If no donors, then use a shared slice

Allocate slice to borrower
Karma allocation algorithm

Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits
   If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor
Karma allocation algorithm

Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits
   If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

<table>
<thead>
<tr>
<th>Credits</th>
<th>Demands</th>
<th>Donated Slices</th>
<th>Shared Slices</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Pick **borrower** with **maximum** credits
2. Pick **donor** with **minimum** credits
3. If no donors, then use a shared slice
4. Allocate slice to borrower
5. -1 credit for borrower, +1 credit for donor
6. (Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- Pick **borrower** with **maximum** credits
- Pick **donor** with **minimum** credits
- If no donors, then use a shared slice
- Allocate slice to borrower
- **-1** credit for borrower, **+1** credit for donor

(Repeat until demands satisfied or resources exhausted)
**Karma allocation algorithm**

**Running Example**

- **Credits**
- **Demands**
- **Donated Slices**
- **Shared Slices**

- **Pick borrower with maximum credits**
- **Pick donor with minimum credits**
  - If no donors, then use a shared slice
- **Allocate slice to borrower**
  - -1 credit for borrower, +1 credit for donor
- (Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

Pick **borrower** with maximum credits

Pick **donor** with minimum credits

If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- Pick **borrower** with **maximum** credits
- Pick **donor** with **minimum** credits
  - If no donors, then use a shared slice
  - Allocate slice to borrower
  - -1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- Pick **borrower** with *maximum* credits
- Pick **donor** with *minimum* credits
  - If no donors, then use a shared slice
  - Allocate slice to borrower
  - -1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

Pick borrower with **maximum** credits

Pick donor with **minimum** credits

If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

1. Pick **borrower** with **maximum** credits.
2. Pick **donor** with **minimum** credits.
3. If no donors, then use a shared slice.
4. Allocate slice to borrower.
5. -1 credit for borrower, +1 credit for donor.
6. (Repeat until demands satisfied or resources exhausted.)
Karma allocation algorithm

Running Example

Pick **borrower** with maximum credits

Pick **donor** with minimum credits

If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- Pick **borrower** with maximum credits

- Pick **donor** with minimum credits
  
  If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- Pick borrower with **maximum** credits
- Pick donor with **minimum** credits
- If no donors, then use a shared slice
- Allocate slice to borrower
- -1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- Pick **borrower** with **maximum** credits
- Pick **donor** with **minimum** credits
  - If no donors, then use a shared slice
  - Allocate slice to borrower
  - -1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- Pick borrower with maximum credits
- Pick donor with minimum credits
- If no donors, then use a shared slice
- Allocate slice to borrower
- -1 credit for borrower, +1 credit for donor
- (Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

Pick borrower with maximum credits

Pick donor with minimum credits
If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor
(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- Pick **borrower** with **maximum** credits
- Pick **donor** with **minimum** credits
- If no donors, then use a shared slice
- Allocate slice to borrower
- -1 credit for borrower, +1 credit for donor
- (Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- Pick **borrower** with *maximum* credits
- Pick **donor** with *minimum* credits
  - If no donors, then use a shared slice
  - Allocate slice to borrower
  - -1 credit for borrower, +1 credit for donor
  - (Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

Pick borrower with **maximum** credits

Pick donor with **minimum** credits
If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

Pick borrower with maximum credits

Pick donor with minimum credits
If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- Pick borrower with **maximum** credits
- Pick donor with **minimum** credits
  - If no donors, then use a shared slice
  - Allocate slice to borrower
  - -1 credit for borrower, +1 credit for donor
    - (Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- **Credits**
  - Borrower with maximum credits
  - Donor with minimum credits

- **Demands**
  - Time 0, 1, 2

- **Donated Slices**
  - Time 0, 1, 2

- **Shared Slices**
  - Time 0, 1, 2

- **Pick borrower with maximum credits**
- **Pick donor with minimum credits**
  - If no donors, then use a shared slice

- **Allocate slice to borrower**
  - -1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- Pick **borrower** with **maximum** credits
- Pick **donor** with **minimum** credits
  - If no donors, then use a shared slice
  - Allocate slice to borrower
  - -1 credit for borrower, +1 credit for donor
  - (Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

<table>
<thead>
<tr>
<th>Credits</th>
<th>Demands</th>
<th>Donated Slices</th>
<th>Shared Slices</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Borrower" /></td>
<td><img src="image2.png" alt="Donor" /></td>
<td><img src="image3.png" alt="Donated Slices" /></td>
<td><img src="image4.png" alt="Shared Slices" /></td>
</tr>
</tbody>
</table>

- Pick **borrower** with maximum credits
- Pick **donor** with minimum credits
  - If no donors, then use a shared slice
  - Allocate slice to borrower
  - $-1$ credit for borrower, $+1$ credit for donor
  - (Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

Pick borrower with maximum credits

Pick donor with minimum credits

If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

<table>
<thead>
<tr>
<th>Credits</th>
<th>Demands</th>
<th>Donated Slices</th>
<th>Shared Slices</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Pick **borrower** with **maximum** credits
- Pick **donor** with **minimum** credits
  - If no donors, then use a shared slice
  - Allocate slice to borrower
  - -1 credit for borrower, +1 credit for donor
  - (Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

Pick borrower with **maximum** credits

Pick donor with **minimum** credits
If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor
(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

Pick borrower with maximum credits

Pick donor with minimum credits

If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- Pick borrower with **maximum** credits
- Pick donor with **minimum** credits
- If no donors, then use a shared slice
- Allocate slice to borrower
- -1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Running Example

Karma allocation algorithm

1. **Pick borrower with maximum credits**
2. **Pick donor with minimum credits**
   - If no donors, then use a shared slice
   - Allocate slice to borrower
   - -1 credit for borrower, +1 credit for donor

   (Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

- **Pick** borrower with maximum credits
- **Pick** donor with minimum credits

If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

Pick borrower with maximum credits

Pick donor with minimum credits
If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

Pick borrower with maximum credits

Pick donor with minimum credits
If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma allocation algorithm

Running Example

<table>
<thead>
<tr>
<th>Credits</th>
<th>Demands</th>
<th>Total Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| - | Credits | Demands | Total Allocation |
| - |         |         |                 |

![Diagram](https://via.placeholder.com/150)
Karma allocation algorithm

Running Example

Credits

Demands

Total Allocation

Equal total allocations!
Karma parameterizes the trade-off between instantaneous and long-term fairness.
Karma parameterizes the trade-off between instantaneous and long-term fairness.
Karma parameterizes the trade-off between instantaneous and long-term fairness.
Karma parameterizes the trade-off between instantaneous and long-term fairness.

(See paper for more details)
Karma: Revisiting the classical resource allocation problem under dynamic demands

New resource allocation algorithm for dynamic demands

Theoretical guarantees

- Pareto efficiency
- Strategy-proofness
- Fairness

Prototype implementation and evaluation
Karma is Pareto efficient

Pareto efficiency
Resources should not remain unused when there is demand
Karma is Pareto efficient

Pareto efficiency
Resources should not remain unused when there is demand

Karma Allocation Algorithm

- Pick **borrower** with **maximum** credits
- Pick **donor** with **minimum** credits
  - If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma is Pareto efficient

**Pareto efficiency**
Resources should not remain unused when there is demand

**Karma Allocation Algorithm**

- Pick **borrower** with **maximum** credits
- Pick **donor** with **minimum** credits
  - If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)
Karma provides powerful strategy-proofness properties

**Strategy-proofness**
Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)
Karma provides powerful strategy-proofness properties

**Strategy-proofness**
Selfish users cannot increase their allocation by lying (selfish ≠ adversarial)
Karma provides powerful strategy-proofness properties

**Strategy-proofness**
Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)

Not possible to increase allocation
Karma provides powerful strategy-proofness properties

**Strategy-proofness**
Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)

Not possible to increase allocation
Karma provides powerful strategy-proofness properties

**Strategy-proofness**
Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)

Not possible to increase allocation

Perfect knowledge of future demands of all users
Karma provides powerful strategy-proofness properties

**Strategy-proofness**
Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)

- Not possible to increase allocation
- Perfect knowledge of future demands of all users
- No more than 1.5x increase in allocation
Karma provides powerful strategy-proofness properties

**Strategy-proofness**
Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)

- **Not possible to increase allocation**
- **Perfect knowledge of future demands of all users**
- **No more than 1.5x increase in allocation**

Far from realistic
Karma provides powerful strategy-proofness properties

**Strategy-proofness**
Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)

- **Perfect knowledge of future demands of all users**
  - Not possible to increase allocation

- **Any imprecision in knowledge of future demands of any user**
  - No more than 1.5x increase in allocation
Karma provides powerful strategy-proofness properties

**Strategy-proofness**
Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)

- Not possible to increase allocation
- Perfect knowledge of future demands of all users
- No more than 1.5x increase in allocation
- Any imprecision in knowledge of future demands of any user
  As much as $\Omega(n)$ factor decrease in allocation
Karma maximizes the minimum total allocation (given past allocations)

Fairness
Balanced resource allocations across users
Karma maximizes the minimum total allocation (given past allocations)

Fairness
Balanced resource allocations across users
Karma maximizes the minimum total allocation (given past allocations)

**Fairness**
Balanced resource allocations across users

Fixed past allocations

Total Allocation (from 0 to t-1)
Karma maximizes the minimum total allocation (given past allocations)

Fairness
Balanced resource allocations across users

Fixed past allocations

Total Allocation (from 0 to t-1)

Credits
Karma: Resource Allocation for Dynamic Demands

New credit-based resource allocation algorithm

Strong theoretical guarantees

Prototype implementation and evaluation
Karma implementation & evaluation
Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys’22])
Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys’22])
Evaluated in-memory key-value cache scenario
Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys’22])
Evaluated in-memory key-value cache scenario

Experimental Setup

EC2 VMs

S3 (Persistent Storage)
Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys’22])
Evaluated in-memory key-value cache scenario

Experimental Setup
Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys’22])
Evaluated in-memory key-value cache scenario

Experimental Setup

EC2 VMs

Jiffy w/ Karma (Cache)

S3 (Persistent Storage)
Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys’22])
Evaluated in-memory key-value cache scenario

Experimental Setup

EC2 VMs

Jiffy w/ Karma (Cache)

S3 (Persistent Storage)
Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys’22])
Evaluated in-memory key-value cache scenario

Experimental Setup

EC2 VMs

Jiffy w/ Karma (Cache)

S3 (Persistent Storage)
Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys’22])
Evaluated in-memory key-value cache scenario

Experimental Setup

Workload
Dynamic demands -> vary working set size
Demands taken from Snowflake dataset
Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys’22])
Evaluated in-memory key-value cache scenario

Experimental Setup

Workload
Dynamic demands -> vary working set size
Demands taken from Snowflake dataset

(See paper for details)
Glimpse of Karma evaluation results
Glimpse of Karma evaluation results

Max-min fairness (Periodic)
Karma

Throughput (kops/sec)

% of users with throughput ≤ x
Glimpse of Karma evaluation results

- Max-min fairness (Periodic)
- Karma

Throughput (kops/sec) vs % of users with throughput ≤ x
Glimpse of Karma evaluation results

Throughput (kops/sec)

% of users with throughput ≤ x

Max-min fairness (Periodic)
Karma
Glimpse of Karma evaluation results

Throughput (kops/sec)

% of users with throughput ≤ x

Max-min fairness (Periodic)
Karma
Glimpse of Karma evaluation results

Throughput (kops/sec)

% of users with throughput ≤ x

Max-min fairness (Periodic)

Karma
Glimpse of Karma evaluation results

Karma minimizes disparity across users
(while maintaining high average performance)
Glimpse of Karma evaluation results

Karma minimizes disparity across users (while maintaining high average performance)

(See paper for more detailed evaluation results)
Karma: Revisiting the classical resource allocation problem under dynamic demands

New resource allocation algorithm for dynamic demands

Theoretical guarantees
- Pareto efficiency
- Strategy-proofness
- Fairness

Prototype implementation and evaluation
Karma: Revisiting the classical resource allocation problem under dynamic demands

New resource allocation algorithm for dynamic demands

Theoretical guarantees

Prototype implementation and evaluation

Karma opens many interesting avenues for future research, both in systems and in theory
Karma: Revisiting the classical resource allocation problem under dynamic demands

New resource allocation algorithm for dynamic demands

Theoretical guarantees

Prototype implementation and evaluation

Karma opens many interesting avenues for future research, both in systems and in theory

https://github.com/resource-disaggregation/karma

Midhul Vuppalapati (midhul@cs.cornell.edu)