Heterogeneity-Aware Hashing

CS:402 Distributed Systems Course Project Presentation

K. Sai Anuroop (170030035) and **Vivek Mishra** (170020007)
Depts. of Computer Science and Engineering and Electrical Engineering
Supervised by: **Prof. Kedar Khandeparkar**
Dept. of Computer Science and Engineering

Indian Institute of Technology Dharwad

December 7, 2020

Abstract

- Hashing techniques are widely used in distributed storage applications.
- In this project, we propose hashing techniques which specifically address the heterogeneity of node capacities.
- We also analyse the proposed techniques in the light of consistent hashing properties and present results of simulations performed based on the proposed techniques.
- Consistent Hashing (CH) maps the *keys* of objects into a dynamic set of *resources* such that there is a minimal disruption to the existing maps when resources are arbitrarily added or removed, and no resource is overloaded.
- CH is widely used in
  1. network load balancers
  2. distributed storage systems
  3. web caching applications
  4. peer to peer (P2P) networks
  5. content delivery networks (CDN)

Abstract

- Hashing techniques are widely used in distributed storage applications.
- In this project, we propose hashing techniques which specifically address the heterogeneity of node capacities.
- We also analyse the proposed techniques in the light of consistent hashing properties and present results of simulations performed based on the proposed techniques.
- Consistent Hashing (CH) maps the *keys* of objects into a dynamic set of *resources* such that there is a minimal disruption to the existing maps when resources are arbitrarily added or removed, and no resource is overloaded.
- CH is widely used in
  1. network load balancers
  2. distributed storage systems
  3. web caching applications
  4. peer to peer (P2P) networks
  5. content delivery networks (CDN)

Abstract

- Hashing techniques are widely used in distributed storage applications.
- In this project, we propose hashing techniques which specifically address the heterogeneity of node capacities.
- We also analyse the proposed techniques in the light of consistent hashing properties and present results of simulations performed based on the proposed techniques.
- Consistent Hashing (CH) maps the *keys* of objects into a dynamic set of *resources* such that there is a minimal disruption to the existing maps when resources are arbitrarily added or removed, and no resource is overloaded.
- CH is widely used in
  1. network load balancers
  2. distributed storage systems
  3. web caching applications
  4. peer to peer (P2P) networks
  5. content delivery networks (CDN)

Abstract

- Hashing techniques are widely used in distributed storage applications.
- In this project, we propose hashing techniques which specifically address the heterogeneity of node capacities.
- We also analyse the proposed techniques in the light of consistent hashing properties and present results of simulations performed based on the proposed techniques.
- Consistent Hashing (CH) maps the *keys* of objects into a dynamic set of *resources* such that there is a minimal disruption to the existing maps when resources are arbitrarily added or removed, and no resource is overloaded.
- CH is widely used in
  1. network load balancers
  2. distributed storage systems
  3. web caching applications
  4. peer to peer (P2P) networks
  5. content delivery networks (CDN)

Abstract

- Hashing techniques are widely used in distributed storage applications.
- In this project, we propose hashing techniques which specifically address the heterogeneity of node capacities.
- We also analyse the proposed techniques in the light of consistent hashing properties and present results of simulations performed based on the proposed techniques.
- Consistent Hashing (CH) maps the *keys* of objects into a dynamic set of *resources* such that there is a minimal disruption to the existing maps when resources are arbitrarily added or removed, and no resource is overloaded.
- CH is widely used in
  1. network load balancers
  2. distributed storage systems
  3. web caching applications
  4. peer to peer (P2P) networks
  5. content delivery networks (CDN)

Abstract

- Hashing techniques are widely used in distributed storage applications.
- In this project, we propose hashing techniques which specifically address the heterogeneity of node capacities.
- We also analyse the proposed techniques in the light of consistent hashing properties and present results of simulations performed based on the proposed techniques.
- Consistent Hashing (CH) maps the *keys* of objects into a dynamic set of *resources* such that there is a minimal disruption to the existing maps when resources are arbitrarily added or removed, and no resource is overloaded.
- CH is widely used in
  1. network load balancers
  2. distributed storage systems
  3. web caching applications
  4. peer to peer (P2P) networks
  5. content delivery networks (CDN)

Abstract

- Hashing techniques are widely used in distributed storage applications.
- In this project, we propose hashing techniques which specifically address the heterogeneity of node capacities.
- We also analyse the proposed techniques in the light of consistent hashing properties and present results of simulations performed based on the proposed techniques.
- Consistent Hashing (CH) maps the *keys* of objects into a dynamic set of *resources* such that there is a minimal disruption to the existing maps when resources are arbitrarily added or removed, and no resource is overloaded.
- CH is widely used in
  1. network load balancers
  2. distributed storage systems
  3. web caching applications
  4. peer to peer (P2P) networks
  5. content delivery networks (CDN)

Abstract

- Hashing techniques are widely used in distributed storage applications.
- In this project, we propose hashing techniques which specifically address the heterogeneity of node capacities.
- We also analyse the proposed techniques in the light of consistent hashing properties and present results of simulations performed based on the proposed techniques.
- Consistent Hashing (CH) maps the *keys* of objects into a dynamic set of *resources* such that there is a minimal disruption to the existing maps when resources are arbitrarily added or removed, and no resource is overloaded.
- CH is widely used in
  1. network load balancers
  2. distributed storage systems
  3. web caching applications
  4. peer to peer (P2P) networks
  5. content delivery networks (CDN)

Abstract

- Hashing techniques are widely used in distributed storage applications.
- In this project, we propose hashing techniques which specifically address the heterogeneity of node capacities.
- We also analyse the proposed techniques in the light of consistent hashing properties and present results of simulations performed based on the proposed techniques.
- Consistent Hashing (CH) maps the *keys* of objects into a dynamic set of *resources* such that there is a minimal disruption to the existing maps when resources are arbitrarily added or removed, and no resource is overloaded.
- CH is widely used in
  1. network load balancers
  2. distributed storage systems
  3. web caching applications
  4. peer to peer (P2P) networks
  5. content delivery networks (CDN)

Abstract

- Hashing techniques are widely used in distributed storage applications.
- In this project, we propose hashing techniques which specifically address the heterogeneity of node capacities.
- We also analyse the proposed techniques in the light of consistent hashing properties and present results of simulations performed based on the proposed techniques.
- Consistent Hashing (CH) maps the *keys* of objects into a dynamic set of *resources* such that there is a minimal disruption to the existing maps when resources are arbitrarily added or removed, and no resource is overloaded.
- CH is widely used in
  1. network load balancers
  2. distributed storage systems
  3. web caching applications
  4. peer to peer (P2P) networks
  5. content delivery networks (CDN)

What are the pitfalls here?

- Most of initial formulations of CH, however are agnostic to the heterogeneity in resource capacities and therefore incorporate uniform hashing functions in their work.
- In this project, we critically analyse one such CH technique called AnchorHash, and highlight some of its lacunae.
- We then propose hashing techniques which specifically address the heterogeneity in resource capacities.

What are the pitfalls here?

- Most of initial formulations of CH, however are agnostic to the heterogeneity in resource capacities and therefore incorporate uniform hashing functions in their work.
- In this project, we critically analyse one such CH technique called AnchorHash, and highlight some of its lacunae.
- We then propose hashing techniques which specifically address the heterogeneity in resource capacities.

What are the pitfalls here?

- Most of initial formulations of CH, however are agnostic to the heterogeneity in resource capacities and therefore incorporate uniform hashing functions in their work.
- In this project, we critically analyse one such CH technique called AnchorHash, and highlight some of its lacunae.
- We then propose hashing techniques which specifically address the heterogeneity in resource capacities.

AnchorHash: Notations

- We wish to map keys to resources.
- Let $\mathcal{U}$ denote the set of keys, and let $\mathcal{S}$ denote the current set of resources.
- We first map keys to *buckets* and then buckets to resources.
- The set of buckets is denoted by $\mathcal{A}$, which is assumed to be finite.
- There exists a bijective mapping between buckets and resources. We henceforth use the terms buckets and resources interchangeably.
- Let $\mathcal{W} \subseteq \mathcal{A}$ denote the subset of buckets that are currently assigned to resources, which are referred to as *working set*.
- Let $\mathcal{W}_b$ denote the working set right after the removal of bucket b.
- Let $\mathcal{R}$ denote the set of removed buckets. Since we are concerned with hashing of the key to a bucket in AnchorHash, we only present that method next.

AnchorHash: Notations

- We wish to map keys to resources.
- Let $\mathcal{U}$ denote the set of keys, and let $\mathcal{S}$ denote the current set of resources.
- We first map keys to *buckets* and then buckets to resources.
- The set of buckets is denoted by $\mathcal{A}$, which is assumed to be finite.
- There exists a bijective mapping between buckets and resources. We henceforth use the terms buckets and resources interchangeably.
- Let $\mathcal{W} \subseteq \mathcal{A}$ denote the subset of buckets that are currently assigned to resources, which are referred to as *working set*.
- Let $\mathcal{W}_b$ denote the working set right after the removal of bucket b.
- Let $\mathcal{R}$ denote the set of removed buckets. Since we are concerned with hashing of the key to a bucket in AnchorHash, we only present that method next.

AnchorHash: Notations

- We wish to map keys to resources.
- Let $\mathcal{U}$ denote the set of keys, and let $\mathcal{S}$ denote the current set of resources.
- We first map keys to *buckets* and then buckets to resources.
- The set of buckets is denoted by $\mathcal{A}$, which is assumed to be finite.
- There exists a bijective mapping between buckets and resources. We henceforth use the terms buckets and resources interchangeably.
- Let $\mathcal{W} \subseteq \mathcal{A}$ denote the subset of buckets that are currently assigned to resources, which are referred to as *working set*.
- Let $\mathcal{W}_b$ denote the working set right after the removal of bucket b.
- Let $\mathcal{R}$ denote the set of removed buckets. Since we are concerned with hashing of the key to a bucket in AnchorHash, we only present that method next.

AnchorHash: Notations

- We wish to map keys to resources.
- Let $\mathcal{U}$ denote the set of keys, and let $\mathcal{S}$ denote the current set of resources.
- We first map keys to *buckets* and then buckets to resources.
- The set of buckets is denoted by $\mathcal{A}$, which is assumed to be finite.
- There exists a bijective mapping between buckets and resources. We henceforth use the terms buckets and resources interchangeably.
- Let $\mathcal{W} \subseteq \mathcal{A}$ denote the subset of buckets that are currently assigned to resources, which are referred to as *working set*.
- Let $\mathcal{W}_b$ denote the working set right after the removal of bucket b.
- Let $\mathcal{R}$ denote the set of removed buckets. Since we are concerned with hashing of the key to a bucket in AnchorHash, we only present that method next.

AnchorHash: Notations

- We wish to map keys to resources.
- Let $\mathcal{U}$ denote the set of keys, and let $\mathcal{S}$ denote the current set of resources.
- We first map keys to *buckets* and then buckets to resources.
- The set of buckets is denoted by $\mathcal{A}$, which is assumed to be finite.
- There exists a bijective mapping between buckets and resources. We henceforth use the terms buckets and resources interchangeably.
- Let $\mathcal{W} \subseteq \mathcal{A}$ denote the subset of buckets that are currently assigned to resources, which are referred to as *working set*.
- Let $\mathcal{W}_b$ denote the working set right after the removal of bucket b.
- Let $\mathcal{R}$ denote the set of removed buckets. Since we are concerned with hashing of the key to a bucket in AnchorHash, we only present that method next.

AnchorHash: Notations

- We wish to map keys to resources.
- Let $\mathcal{U}$ denote the set of keys, and let $\mathcal{S}$ denote the current set of resources.
- We first map keys to *buckets* and then buckets to resources.
- The set of buckets is denoted by $\mathcal{A}$, which is assumed to be finite.
- There exists a bijective mapping between buckets and resources. We henceforth use the terms buckets and resources interchangeably.
- Let $\mathcal{W} \subseteq \mathcal{A}$ denote the subset of buckets that are currently assigned to resources, which are referred to as *working set*.
- Let $\mathcal{W}_b$ denote the working set right after the removal of bucket b.
- Let $\mathcal{R}$ denote the set of removed buckets. Since we are concerned with hashing of the key to a bucket in AnchorHash, we only present that method next.

AnchorHash: Notations

- We wish to map keys to resources.
- Let $\mathcal{U}$ denote the set of keys, and let $\mathcal{S}$ denote the current set of resources.
- We first map keys to *buckets* and then buckets to resources.
- The set of buckets is denoted by $\mathcal{A}$, which is assumed to be finite.
- There exists a bijective mapping between buckets and resources. We henceforth use the terms buckets and resources interchangeably.
- Let $\mathcal{W} \subseteq \mathcal{A}$ denote the subset of buckets that are currently assigned to resources, which are referred to as *working set*.
- Let $\mathcal{W}_b$ denote the working set right after the removal of bucket b.
- Let $\mathcal{R}$ denote the set of removed buckets. Since we are concerned with hashing of the key to a bucket in AnchorHash, we only present that method next.

AnchorHash: Notations

- We wish to map keys to resources.
- Let $\mathcal{U}$ denote the set of keys, and let $\mathcal{S}$ denote the current set of resources.
- We first map keys to *buckets* and then buckets to resources.
- The set of buckets is denoted by $\mathcal{A}$, which is assumed to be finite.
- There exists a bijective mapping between buckets and resources. We henceforth use the terms buckets and resources interchangeably.
- Let $\mathcal{W} \subseteq \mathcal{A}$ denote the subset of buckets that are currently assigned to resources, which are referred to as *working set*.
- Let $\mathcal{W}_b$ denote the working set right after the removal of bucket b.
- Let $\mathcal{R}$ denote the set of removed buckets. Since we are concerned with hashing of the key to a bucket in AnchorHash, we only present that method next.

AnchorHash: Hashing

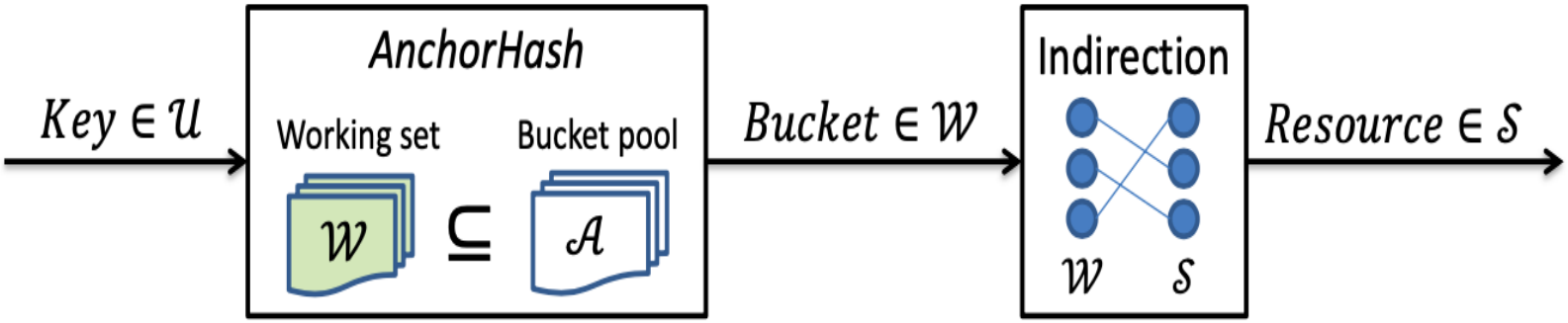

Fig. 1: AnchorHash uses indirection in order to compute the key-to-resource mapping. It first sets a bijective mapping between buckets and resources (right side), and then computes for each incoming key a key-to-bucket mapping (left side).

Figure: Technique AnchorHash*

*https://arxiv.org/pdf/1812.09674.pdf

Drawbacks of AnchorHash in a heterogeneous resource environment

- Consider a partition of $\mathcal{S}$ into $\mathcal{S}_{high}$ and $\mathcal{S}_{low}$ such that the resources $s \in \mathcal{S}_{high}$ have a metric $M_s \geq M_{s'} \ \forall \ s' \in \mathcal{S}_{low}$.
- Let $|\mathcal{W}| = w$, $|\mathcal{W}_S| = w_s$ and $|\mathcal{W}_{S'}| = w_{s'}$.
- $m_s = \min_{s \in \mathcal{W}_S} M_s$ and $m_{s'} = \min_{s' \in \mathcal{W}_{S'}} M_{s'}$
- Let us further assume that more keys can be assigned to a resource with a higher metric than to a resource with a lower metric.
- An optimal allocation would be one where more keys are assigned to a resource with higher metric than to a resource with a lower metric.
- AnchorHash, however, would allocate $\frac{|\mathcal{U}|}{w}$ keys per bucket, which is clearly not optimal in terms of sharing load.
- Another feature of AnchorHash is that it adds the last removed bucket to $\mathcal{W}$.
- Instead, during the addition of the last removed bucket back to $\mathcal{W}$, preference should be given to pair the bucket with that resource which belongs to S_{high} and not in $\mathcal{W}$. If not, the resources with higher capacity may starve, leading to an increased load on the resources with a lower capacity which are already in the working set $\mathcal{W}$.

Drawbacks of AnchorHash in a heterogeneous resource environment

- Consider a partition of $\mathcal{S}$ into $\mathcal{S}_{high}$ and $\mathcal{S}_{low}$ such that the resources $s \in \mathcal{S}_{high}$ have a metric $M_s \geq M_{s'} \ \forall \ s' \in \mathcal{S}_{low}$.
- Let $|\mathcal{W}| = w$, $|\mathcal{W}_S| = w_s$ and $|\mathcal{W}_{S'}| = w_{s'}$.
- $m_s = \min_{s \in \mathcal{W}_S} M_s$ and $m_{s'} = \min_{s' \in \mathcal{W}_{S'}} M_{s'}$
- Let us further assume that more keys can be assigned to a resource with a higher metric than to a resource with a lower metric.
- An optimal allocation would be one where more keys are assigned to a resource with higher metric than to a resource with a lower metric.
- AnchorHash, however, would allocate $\frac{|\mathcal{U}|}{w}$ keys per bucket, which is clearly not optimal in terms of sharing load.
- Another feature of AnchorHash is that it adds the last removed bucket to $\mathcal{W}$.
- Instead, during the addition of the last removed bucket back to $\mathcal{W}$, preference should be given to pair the bucket with that resource which belongs to S_{high} and not in $\mathcal{W}$. If not, the resources with higher capacity may starve, leading to an increased load on the resources with a lower capacity which are already in the working set $\mathcal{W}$.

Drawbacks of AnchorHash in a heterogeneous resource environment

- Consider a partition of $\mathcal{S}$ into $\mathcal{S}_{high}$ and $\mathcal{S}_{low}$ such that the resources $s \in \mathcal{S}_{high}$ have a metric $M_s \geq M_{s'} \ \forall \ s' \in \mathcal{S}_{low}$.
- Let $|\mathcal{W}| = w$, $|\mathcal{W}_S| = w_s$ and $|\mathcal{W}_{S'}| = w_{s'}$.
- $m_s = \min_{s \in \mathcal{W}_S} M_s$ and $m_{s'} = \min_{s' \in \mathcal{W}_{S'}} M_{s'}$
- Let us further assume that more keys can be assigned to a resource with a higher metric than to a resource with a lower metric.
- An optimal allocation would be one where more keys are assigned to a resource with higher metric than to a resource with a lower metric.
- AnchorHash, however, would allocate $\frac{|\mathcal{U}|}{w}$ keys per bucket, which is clearly not optimal in terms of sharing load.
- Another feature of AnchorHash is that it adds the last removed bucket to $\mathcal{W}$.
- Instead, during the addition of the last removed bucket back to $\mathcal{W}$, preference should be given to pair the bucket with that resource which belongs to S_{high} and not in $\mathcal{W}$. If not, the resources with higher capacity may starve, leading to an increased load on the resources with a lower capacity which are already in the working set $\mathcal{W}$.

Drawbacks of AnchorHash in a heterogeneous resource environment

- Consider a partition of $\mathcal{S}$ into $\mathcal{S}_{high}$ and $\mathcal{S}_{low}$ such that the resources $s \in \mathcal{S}_{high}$ have a metric $M_s \geq M_{s'} \ \forall \ s' \in \mathcal{S}_{low}$.
- Let $|\mathcal{W}| = w$, $|\mathcal{W}_S| = w_s$ and $|\mathcal{W}_{S'}| = w_{s'}$.
- $m_s = \min_{s \in \mathcal{W}_S} M_s$ and $m_{s'} = \min_{s' \in \mathcal{W}_{S'}} M_{s'}$
- Let us further assume that more keys can be assigned to a resource with a higher metric than to a resource with a lower metric.
- An optimal allocation would be one where more keys are assigned to a resource with higher metric than to a resource with a lower metric.
- AnchorHash, however, would allocate $\frac{|\mathcal{U}|}{w}$ keys per bucket, which is clearly not optimal in terms of sharing load.
- Another feature of AnchorHash is that it adds the last removed bucket to $\mathcal{W}$.
- Instead, during the addition of the last removed bucket back to $\mathcal{W}$, preference should be given to pair the bucket with that resource which belongs to S_{high} and not in $\mathcal{W}$. If not, the resources with higher capacity may starve, leading to an increased load on the resources with a lower capacity which are already in the working set $\mathcal{W}$.

Drawbacks of AnchorHash in a heterogeneous resource environment

- Consider a partition of $\mathcal{S}$ into $\mathcal{S}_{high}$ and $\mathcal{S}_{low}$ such that the resources $s \in \mathcal{S}_{high}$ have a metric $M_s \geq M_{s'} \ \forall \ s' \in \mathcal{S}_{low}$.
- Let $|\mathcal{W}| = w$, $|\mathcal{W}_S| = w_s$ and $|\mathcal{W}_{S'}| = w_{s'}$.
- $m_s = \min_{s \in \mathcal{W}_S} M_s$ and $m_{s'} = \min_{s' \in \mathcal{W}_{S'}} M_{s'}$
- Let us further assume that more keys can be assigned to a resource with a higher metric than to a resource with a lower metric.
- An optimal allocation would be one where more keys are assigned to a resource with higher metric than to a resource with a lower metric.
- AnchorHash, however, would allocate $\frac{|\mathcal{U}|}{w}$ keys per bucket, which is clearly not optimal in terms of sharing load.
- Another feature of AnchorHash is that it adds the last removed bucket to $\mathcal{W}$.
- Instead, during the addition of the last removed bucket back to $\mathcal{W}$, preference should be given to pair the bucket with that resource which belongs to S_{high} and not in $\mathcal{W}$. If not, the resources with higher capacity may starve, leading to an increased load on the resources with a lower capacity which are already in the working set $\mathcal{W}$.

Drawbacks of AnchorHash in a heterogeneous resource environment

- Consider a partition of $\mathcal{S}$ into $\mathcal{S}_{high}$ and $\mathcal{S}_{low}$ such that the resources $s \in \mathcal{S}_{high}$ have a metric $M_s \geq M_{s'} \ \forall \ s' \in \mathcal{S}_{low}$.
- Let $|\mathcal{W}| = w$, $|\mathcal{W}_S| = w_s$ and $|\mathcal{W}_{S'}| = w_{s'}$.
- $m_s = \min_{s \in \mathcal{W}_S} M_s$ and $m_{s'} = \min_{s' \in \mathcal{W}_{S'}} M_{s'}$
- Let us further assume that more keys can be assigned to a resource with a higher metric than to a resource with a lower metric.
- An optimal allocation would be one where more keys are assigned to a resource with higher metric than to a resource with a lower metric.
- AnchorHash, however, would allocate $\frac{|\mathcal{U}|}{w}$ keys per bucket, which is clearly not optimal in terms of sharing load.
- Another feature of AnchorHash is that it adds the last removed bucket to $\mathcal{W}$.
- Instead, during the addition of the last removed bucket back to $\mathcal{W}$, preference should be given to pair the bucket with that resource which belongs to S_{high} and not in $\mathcal{W}$. If not, the resources with higher capacity may starve, leading to an increased load on the resources with a lower capacity which are already in the working set $\mathcal{W}$.

Drawbacks of AnchorHash in a heterogeneous resource environment

- Consider a partition of $\mathcal{S}$ into $\mathcal{S}_{high}$ and $\mathcal{S}_{low}$ such that the resources $s \in \mathcal{S}_{high}$ have a metric $M_s \geq M_{s'} \ \forall \ s' \in \mathcal{S}_{low}$.
- Let $|\mathcal{W}| = w$, $|\mathcal{W}_S| = w_s$ and $|\mathcal{W}_{S'}| = w_{s'}$.
- $m_s = \min_{s \in \mathcal{W}_S} M_s$ and $m_{s'} = \min_{s' \in \mathcal{W}_{S'}} M_{s'}$
- Let us further assume that more keys can be assigned to a resource with a higher metric than to a resource with a lower metric.
- An optimal allocation would be one where more keys are assigned to a resource with higher metric than to a resource with a lower metric.
- AnchorHash, however, would allocate $\frac{|\mathcal{U}|}{w}$ keys per bucket, which is clearly not optimal in terms of sharing load.
- Another feature of AnchorHash is that it adds the last removed bucket to $\mathcal{W}$.
- Instead, during the addition of the last removed bucket back to $\mathcal{W}$, preference should be given to pair the bucket with that resource which belongs to S_{high} and not in $\mathcal{W}$. If not, the resources with higher capacity may starve, leading to an increased load on the resources with a lower capacity which are already in the working set $\mathcal{W}$.

Drawbacks of AnchorHash in a heterogeneous resource environment

- Consider a partition of $\mathcal{S}$ into $\mathcal{S}_{high}$ and $\mathcal{S}_{low}$ such that the resources $s \in \mathcal{S}_{high}$ have a metric $M_s \geq M_{s'} \ \forall \ s' \in \mathcal{S}_{low}$.
- Let $|\mathcal{W}| = w$, $|\mathcal{W}_S| = w_s$ and $|\mathcal{W}_{S'}| = w_{s'}$.
- $m_s = \min_{s \in \mathcal{W}_S} M_s$ and $m_{s'} = \min_{s' \in \mathcal{W}_{S'}} M_{s'}$
- Let us further assume that more keys can be assigned to a resource with a higher metric than to a resource with a lower metric.
- An optimal allocation would be one where more keys are assigned to a resource with higher metric than to a resource with a lower metric.
- AnchorHash, however, would allocate $\frac{|\mathcal{U}|}{w}$ keys per bucket, which is clearly not optimal in terms of sharing load.
- Another feature of AnchorHash is that it adds the last removed bucket to $\mathcal{W}$.
- Instead, during the addition of the last removed bucket back to $\mathcal{W}$, preference should be given to pair the bucket with that resource which belongs to S_{high} and not in $\mathcal{W}$. If not, the resources with higher capacity may starve, leading to an increased load on the resources with a lower capacity which are already in the working set $\mathcal{W}$.

Heterogeneity-Aware Hashing

- Let us assume the following:
  1. Resources (buckets) are hashed to the ring first.
  2. Incoming keys are then hashed to the ring.
  3. A bucket is responsible for all the keys that are hashed to the positions on the ring, starting from the position to which the bucket is hashed, up to the position (excluding) to which the next bucket is hashed, in counter-clockwise direction.
- We now propose two heterogeneity-aware hashing techniques:
  1. the first one takes into account the excess of capacity available with a subset of resources and does the mapping of resources to the positions on the ring accordingly.
  2. the second considers the capacity of each resource with respect to the minimum capacity among all nodes and does the mapping of resources to the positions on the ring accordingly

Heterogeneity-Aware Hashing

- Let us assume the following:
  1. Resources (buckets) are hashed to the ring first.
  2. Incoming keys are then hashed to the ring.
  3. A bucket is responsible for all the keys that are hashed to the positions on the ring, starting from the position to which the bucket is hashed, up to the position (excluding) to which the next bucket is hashed, in counter-clockwise direction.
- We now propose two heterogeneity-aware hashing techniques:
  1. the first one takes into account the excess of capacity available with a subset of resources and does the mapping of resources to the positions on the ring accordingly.
  2. the second considers the capacity of each resource with respect to the minimum capacity among all nodes and does the mapping of resources to the positions on the ring accordingly

Heterogeneity-Aware Hashing

- Let us assume the following:
  1. Resources (buckets) are hashed to the ring first.
  2. Incoming keys are then hashed to the ring.
  3. A bucket is responsible for all the keys that are hashed to the positions on the ring, starting from the position to which the bucket is hashed, up to the position (excluding) to which the next bucket is hashed, in counter-clockwise direction.
- We now propose two heterogeneity-aware hashing techniques:
  1. the first one takes into account the excess of capacity available with a subset of resources and does the mapping of resources to the positions on the ring accordingly.
  2. the second considers the capacity of each resource with respect to the minimum capacity among all nodes and does the mapping of resources to the positions on the ring accordingly

Heterogeneity-Aware Hashing

- Let us assume the following:
  1. Resources (buckets) are hashed to the ring first.
  2. Incoming keys are then hashed to the ring.
  3. A bucket is responsible for all the keys that are hashed to the positions on the ring, starting from the position to which the bucket is hashed, up to the position (excluding) to which the next bucket is hashed, in counter-clockwise direction.
- We now propose two heterogeneity-aware hashing techniques:
  1. the first one takes into account the excess of capacity available with a subset of resources and does the mapping of resources to the positions on the ring accordingly.
  2. the second considers the capacity of each resource with respect to the minimum capacity among all nodes and does the mapping of resources to the positions on the ring accordingly

Heterogeneity-Aware Hashing

- Let us assume the following:
  1. Resources (buckets) are hashed to the ring first.
  2. Incoming keys are then hashed to the ring.
  3. A bucket is responsible for all the keys that are hashed to the positions on the ring, starting from the position to which the bucket is hashed, up to the position (excluding) to which the next bucket is hashed, in counter-clockwise direction.
- We now propose two heterogeneity-aware hashing techniques:
  1. the first one takes into account the excess of capacity available with a subset of resources and does the mapping of resources to the positions on the ring accordingly.
  2. the second considers the capacity of each resource with respect to the minimum capacity among all nodes and does the mapping of resources to the positions on the ring accordingly

Heterogeneity-Aware Hashing

- Let us assume the following:
  1. Resources (buckets) are hashed to the ring first.
  2. Incoming keys are then hashed to the ring.
  3. A bucket is responsible for all the keys that are hashed to the positions on the ring, starting from the position to which the bucket is hashed, up to the position (excluding) to which the next bucket is hashed, in counter-clockwise direction.
- We now propose two heterogeneity-aware hashing techniques:
  1. the first one takes into account the excess of capacity available with a subset of resources and does the mapping of resources to the positions on the ring accordingly.
  2. the second considers the capacity of each resource with respect to the minimum capacity among all nodes and does the mapping of resources to the positions on the ring accordingly

Heterogeneity-Aware Hashing

- Let us assume the following:
  1. Resources (buckets) are hashed to the ring first.
  2. Incoming keys are then hashed to the ring.
  3. A bucket is responsible for all the keys that are hashed to the positions on the ring, starting from the position to which the bucket is hashed, up to the position (excluding) to which the next bucket is hashed, in counter-clockwise direction.
- We now propose two heterogeneity-aware hashing techniques:
  1. the first one takes into account the excess of capacity available with a subset of resources and does the mapping of resources to the positions on the ring accordingly.
  2. the second considers the capacity of each resource with respect to the minimum capacity among all nodes and does the mapping of resources to the positions on the ring accordingly

Heterogeneity-Aware Hashing

- Let us assume the following:
  1. Resources (buckets) are hashed to the ring first.
  2. Incoming keys are then hashed to the ring.
  3. A bucket is responsible for all the keys that are hashed to the positions on the ring, starting from the position to which the bucket is hashed, up to the position (excluding) to which the next bucket is hashed, in counter-clockwise direction.
- We now propose two heterogeneity-aware hashing techniques:
  1. the first one takes into account the excess of capacity available with a subset of resources and does the mapping of resources to the positions on the ring accordingly.
  2. the second considers the capacity of each resource with respect to the minimum capacity among all nodes and does the mapping of resources to the positions on the ring accordingly

Single threshold based hashing (Thresh-Hash)

- Notice that the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$ have a net higher capacity of $w_s m_s - w_{s'} m_{s'}$ than those in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{low}$.
- This extra capacity has to be distributed evenly among the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$.
- So, $\frac{w_s m_s - w_{s'} m_{s'}}{w_s}$ is the extra capacity that is available at each of the resource in the working set $\mathcal{W}$ which belongs to $\mathcal{S}_{high}$.
- This means that we need to hash a resource belonging to $\mathcal{S}_{high}$ to the ring for a total of $1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil$ times if this value is greater than 1 or else we hash only for 1 time.
- We make use of multiple hash functions which perform this job for us.
- For mapping between buckets and resources, we suggest a greedy approach where the resource with the highest metric, if available, is paired with the bucket being added to the working set.

Single threshold based hashing (Thresh-Hash)

- Notice that the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$ have a net higher capacity of $w_s m_s - w_{s'} m_{s'}$ than those in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{low}$.
- This extra capacity has to be distributed evenly among the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$.
- So, $\frac{w_s m_s - w_{s'} m_{s'}}{w_s}$ is the extra capacity that is available at each of the resource in the working set $\mathcal{W}$ which belongs to $\mathcal{S}_{high}$.
- This means that we need to hash a resource belonging to $\mathcal{S}_{high}$ to the ring for a total of $1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil$ times if this value is greater than 1 or else we hash only for 1 time.
- We make use of multiple hash functions which perform this job for us.
- For mapping between buckets and resources, we suggest a greedy approach where the resource with the highest metric, if available, is paired with the bucket being added to the working set.

Single threshold based hashing (Thresh-Hash)

- Notice that the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$ have a net higher capacity of $w_s m_s - w_{s'} m_{s'}$ than those in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{low}$.
- This extra capacity has to be distributed evenly among the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$.
- So, $\frac{w_s m_s - w_{s'} m_{s'}}{w_s}$ is the extra capacity that is available at each of the resource in the working set $\mathcal{W}$ which belongs to $\mathcal{S}_{high}$.
- This means that we need to hash a resource belonging to $\mathcal{S}_{high}$ to the ring for a total of $1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil$ times if this value is greater than 1 or else we hash only for 1 time.
- We make use of multiple hash functions which perform this job for us.
- For mapping between buckets and resources, we suggest a greedy approach where the resource with the highest metric, if available, is paired with the bucket being added to the working set.

Single threshold based hashing (Thresh-Hash)

- Notice that the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$ have a net higher capacity of $w_s m_s - w_{s'} m_{s'}$ than those in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{low}$.
- This extra capacity has to be distributed evenly among the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$.
- So, $\frac{w_s m_s - w_{s'} m_{s'}}{w_s}$ is the extra capacity that is available at each of the resource in the working set $\mathcal{W}$ which belongs to $\mathcal{S}_{high}$.
- This means that we need to hash a resource belonging to $\mathcal{S}_{high}$ to the ring for a total of $1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil$ times if this value is greater than 1 or else we hash only for 1 time.
- We make use of multiple hash functions which perform this job for us.
- For mapping between buckets and resources, we suggest a greedy approach where the resource with the highest metric, if available, is paired with the bucket being added to the working set.

Single threshold based hashing (Thresh-Hash)

- Notice that the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$ have a net higher capacity of $w_s m_s - w_{s'} m_{s'}$ than those in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{low}$.
- This extra capacity has to be distributed evenly among the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$.
- So, $\frac{w_s m_s - w_{s'} m_{s'}}{w_s}$ is the extra capacity that is available at each of the resource in the working set $\mathcal{W}$ which belongs to $\mathcal{S}_{high}$.
- This means that we need to hash a resource belonging to $\mathcal{S}_{high}$ to the ring for a total of $1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil$ times if this value is greater than 1 or else we hash only for 1 time.
- We make use of multiple hash functions which perform this job for us.
- For mapping between buckets and resources, we suggest a greedy approach where the resource with the highest metric, if available, is paired with the bucket being added to the working set.

Single threshold based hashing (Thresh-Hash)

- Notice that the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$ have a net higher capacity of $w_s m_s - w_{s'} m_{s'}$ than those in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{low}$.
- This extra capacity has to be distributed evenly among the resources in the working set $\mathcal{W}$ which belong to $\mathcal{S}_{high}$.
- So, $\frac{w_s m_s - w_{s'} m_{s'}}{w_s}$ is the extra capacity that is available at each of the resource in the working set $\mathcal{W}$ which belongs to $\mathcal{S}_{high}$.
- This means that we need to hash a resource belonging to $\mathcal{S}_{high}$ to the ring for a total of $1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil$ times if this value is greater than 1 or else we hash only for 1 time.
- We make use of multiple hash functions which perform this job for us.
- For mapping between buckets and resources, we suggest a greedy approach where the resource with the highest metric, if available, is paired with the bucket being added to the working set.

Hashing using different bucket sizes (Buck-Hash)

- Let $M_{min} = \min_{s \in \mathcal{S}} M_s$.
- We then divide the capacity of each resource $s \in \mathcal{S}$ by M_{min} and take the ceiling of it. This is the number of buckets we assign to s.

  In other words, $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ is the number of buckets assigned to s.

- We hash the resource s to the ring for $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ times using multiple hash functions.
- Notice that in this technique, we assign buckets to the resources in multiples of the lowest capacity among all the resources.
- Accordingly, resources with higher capacities get more buckets, i.e., these resources get mapped to more number of positions on the ring than the ones with lower capacities, and hence they get a greater share of keys.

In the following section, we analyse our proposed techniques in the light of some CH properties.

Hashing using different bucket sizes (Buck-Hash)

- Let $M_{min} = \min_{s \in \mathcal{S}} M_s$.
- We then divide the capacity of each resource $s \in \mathcal{S}$ by M_{min} and take the ceiling of it. This is the number of buckets we assign to s.

  In other words, $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ is the number of buckets assigned to s.

- We hash the resource s to the ring for $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ times using multiple hash functions.
- Notice that in this technique, we assign buckets to the resources in multiples of the lowest capacity among all the resources.
- Accordingly, resources with higher capacities get more buckets, i.e., these resources get mapped to more number of positions on the ring than the ones with lower capacities, and hence they get a greater share of keys.

In the following section, we analyse our proposed techniques in the light of some CH properties.

Hashing using different bucket sizes (Buck-Hash)

- Let $M_{min} = \min_{s \in \mathcal{S}} M_s$.
- We then divide the capacity of each resource $s \in \mathcal{S}$ by M_{min} and take the ceiling of it. This is the number of buckets we assign to s.

  In other words, $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ is the number of buckets assigned to s.
- We hash the resource s to the ring for $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ times using multiple hash functions.
- Notice that in this technique, we assign buckets to the resources in multiples of the lowest capacity among all the resources.
- Accordingly, resources with higher capacities get more buckets, i.e., these resources get mapped to more number of positions on the ring than the ones with lower capacities, and hence they get a greater share of keys.

In the following section, we analyse our proposed techniques in the light of some CH properties.

Hashing using different bucket sizes (Buck-Hash)

- Let $M_{min} = \min_{s \in \mathcal{S}} M_s$.
- We then divide the capacity of each resource $s \in \mathcal{S}$ by M_{min} and take the ceiling of it. This is the number of buckets we assign to s.

  In other words, $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ is the number of buckets assigned to s.
- We hash the resource s to the ring for $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ times using multiple hash functions.
- Notice that in this technique, we assign buckets to the resources in multiples of the lowest capacity among all the resources.
- Accordingly, resources with higher capacities get more buckets, i.e., these resources get mapped to more number of positions on the ring than the ones with lower capacities, and hence they get a greater share of keys.

In the following section, we analyse our proposed techniques in the light of some CH properties.

Hashing using different bucket sizes (Buck-Hash)

- Let $M_{min} = \min_{s \in \mathcal{S}} M_s$.
- We then divide the capacity of each resource $s \in \mathcal{S}$ by M_{min} and take the ceiling of it. This is the number of buckets we assign to s.

  In other words, $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ is the number of buckets assigned to s.
- We hash the resource s to the ring for $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ times using multiple hash functions.
- Notice that in this technique, we assign buckets to the resources in multiples of the lowest capacity among all the resources.
- Accordingly, resources with higher capacities get more buckets, i.e., these resources get mapped to more number of positions on the ring than the ones with lower capacities, and hence they get a greater share of keys.

In the following section, we analyse our proposed techniques in the light of some CH properties.

Hashing using different bucket sizes (Buck-Hash)

- Let $M_{min} = \min_{s \in \mathcal{S}} M_s$.
- We then divide the capacity of each resource $s \in \mathcal{S}$ by M_{min} and take the ceiling of it. This is the number of buckets we assign to s.

  In other words, $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ is the number of buckets assigned to s.
- We hash the resource s to the ring for $\left\lceil \frac{M_s}{M_{min}} \right\rceil$ times using multiple hash functions.
- Notice that in this technique, we assign buckets to the resources in multiples of the lowest capacity among all the resources.
- Accordingly, resources with higher capacities get more buckets, i.e., these resources get mapped to more number of positions on the ring than the ones with lower capacities, and hence they get a greater share of keys.

In the following section, we analyse our proposed techniques in the light of some CH properties.

Properties

We define some properties that a hashing technique has to satisfy in order to be *consistent*.

- Minimal Disruption
  - A hash algorithm achieves minimal disruption if and only if
    1. Upon the addition of a bucket $b \in \mathcal{W}$ to $\mathcal{W}$, keys either maintain their mapping or are remapped to b.
    2. Upon the removal of a bucket $b \in \mathcal{W}$, keys that were not mapped to b keep their mapping. Keys that were mapped to b are remapped to members of $\mathcal{W} \setminus b$
  - We note here that our proposals do not influence the addition or removal of buckets to $\mathcal{W}$.
  - Since AnchorHash satisifies *Minimal Disruption*, so do our proposals Thresh-Hash and Buck-Hash which are merely additions to AnchorHash.

Properties

We define some properties that a hashing technique has to satisfy in order to be *consistent*.

- Minimal Disruption
  - A hash algorithm achieves minimal disruption if and only if
    1. Upon the addition of a bucket $b \in \mathcal{W}$ to W, keys either maintain their mapping or are remapped to b.
    2. Upon the removal of a bucket $b \in \mathcal{W}$, keys that were not mapped to b keep their mapping. Keys that were mapped to b are remapped to members of $\mathcal{W} \setminus b$
  - We note here that our proposals do not influence the addition or removal of buckets to $\mathcal{W}$.
  - Since AnchorHash satisifies *Minimal Disruption*, so do our proposals Thresh-Hash and Buck-Hash which are merely additions to AnchorHash.

Properties

We define some properties that a hashing technique has to satisfy in order to be *consistent*.

- Minimal Disruption
  - A hash algorithm achieves minimal disruption if and only if
    1. Upon the addition of a bucket $b \in \mathcal{W}$ to W, keys either maintain their mapping or are remapped to b.
    2. Upon the removal of a bucket $b \in \mathcal{W}$, keys that were not mapped to b keep their mapping. Keys that were mapped to b are remapped to members of $\mathcal{W} \setminus b$
  - We note here that our proposals do not influence the addition or removal of buckets to $\mathcal{W}$.
  - Since AnchorHash satisifies *Minimal Disruption*, so do our proposals Thresh-Hash and Buck-Hash which are merely additions to AnchorHash.

Properties

We define some properties that a hashing technique has to satisfy in order to be *consistent*.

- Minimal Disruption
  - A hash algorithm achieves minimal disruption if and only if
    1. Upon the addition of a bucket $b \in \mathcal{W}$ to W, keys either maintain their mapping or are remapped to b.
    2. Upon the removal of a bucket $b \in \mathcal{W}$, keys that were not mapped to b keep their mapping. Keys that were mapped to b are remapped to members of $\mathcal{W} \setminus b$
  - We note here that our proposals do not influence the addition or removal of buckets to $\mathcal{W}$.
  - Since AnchorHash satisifies *Minimal Disruption*, so do our proposals Thresh-Hash and Buck-Hash which are merely additions to AnchorHash.

Properties

We define some properties that a hashing technique has to satisfy in order to be *consistent*.

- Minimal Disruption
  - A hash algorithm achieves minimal disruption if and only if
    1. Upon the addition of a bucket $b \in \mathcal{W}$ to W, keys either maintain their mapping or are remapped to b.
    2. Upon the removal of a bucket $b \in \mathcal{W}$, keys that were not mapped to b keep their mapping. Keys that were mapped to b are remapped to members of $\mathcal{W} \setminus b$
  - We note here that our proposals do not influence the addition or removal of buckets to $\mathcal{W}$.
  - Since AnchorHash satisifies *Minimal Disruption*, so do our proposals Thresh-Hash and Buck-Hash which are merely additions to AnchorHash.

Properties

We define some properties that a hashing technique has to satisfy in order to be *consistent*.

- Minimal Disruption
  - A hash algorithm achieves minimal disruption if and only if
    1. Upon the addition of a bucket $b \in \mathcal{W}$ to W, keys either maintain their mapping or are remapped to b.
    2. Upon the removal of a bucket $b \in \mathcal{W}$, keys that were not mapped to b keep their mapping. Keys that were mapped to b are remapped to members of $\mathcal{W} \setminus b$
  - We note here that our proposals do not influence the addition or removal of buckets to $\mathcal{W}$.
  - Since AnchorHash satisifies *Minimal Disruption*, so do our proposals Thresh-Hash and Buck-Hash which are merely additions to AnchorHash.

Properties

- Balance
  - Let $k \in \mathcal{U}$ be a key, chosen uniformly at random.
  - A hash algorithm achieves balance if and only if k has an equal probability of being mapped to each bucket in $\mathcal{W}$.
  - We note that our proposals are orthogonal to the property of *Balance*.
  - Since the resources are mapped to the ring multiple times as per the techniques outlined in Thresh-Hash and Buck-Hash, the mapping is no longer uniform anymore.
  - A key $k \in \mathcal{U}$ has a probability of $\frac{1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil}{P}$ for being mapped to a resource in $\mathcal{S}_{high}$ as per technique Thresh-Hash and a probability of $\frac{\left\lceil \frac{M_s}{M_{min}} \right\rceil}{P}$ for being mapped to a resource as per technique Buck-Hash, where P is the total number of positions on the ring.

Properties

- Balance
  - Let $k \in \mathcal{U}$ be a key, chosen uniformly at random.
  - A hash algorithm achieves balance if and only if k has an equal probability of being mapped to each bucket in $\mathcal{W}$.
  - We note that our proposals are orthogonal to the property of *Balance*.
  - Since the resources are mapped to the ring multiple times as per the techniques outlined in Thresh-Hash and Buck-Hash, the mapping is no longer uniform anymore.
  - A key $k \in \mathcal{U}$ has a probability of $\frac{1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil}{P}$ for being mapped to a resource in $\mathcal{S}_{high}$ as per technique Thresh-Hash and a probability of $\frac{\left\lceil \frac{M_s}{M_{min}} \right\rceil}{P}$ for being mapped to a resource as per technique Buck-Hash, where P is the total number of positions on the ring.

Properties

- Balance
  - Let $k \in \mathcal{U}$ be a key, chosen uniformly at random.
  - A hash algorithm achieves balance if and only if k has an equal probability of being mapped to each bucket in $\mathcal{W}$.
  - We note that our proposals are orthogonal to the property of *Balance*.
  - Since the resources are mapped to the ring multiple times as per the techniques outlined in Thresh-Hash and Buck-Hash, the mapping is no longer uniform anymore.
  - A key $k \in \mathcal{U}$ has a probability of $\frac{1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil}{P}$ for being mapped to a resource in $\mathcal{S}_{high}$ as per technique Thresh-Hash and a probability of $\frac{\left\lceil \frac{M_s}{M_{min}} \right\rceil}{P}$ for being mapped to a resource as per technique Buck-Hash, where P is the total number of positions on the ring.

Properties

- Balance
  - Let $k \in \mathcal{U}$ be a key, chosen uniformly at random.
  - A hash algorithm achieves balance if and only if k has an equal probability of being mapped to each bucket in $\mathcal{W}$.
  - We note that our proposals are orthogonal to the property of *Balance*.
  - Since the resources are mapped to the ring multiple times as per the techniques outlined in Thresh-Hash and Buck-Hash, the mapping is no longer uniform anymore.
  - A key $k \in \mathcal{U}$ has a probability of $\frac{1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil}{P}$ for being mapped to a resource in $\mathcal{S}_{high}$ as per technique Thresh-Hash and a probability of $\frac{\left\lceil \frac{M_s}{M_{min}} \right\rceil}{P}$ for being mapped to a resource as per technique Buck-Hash, where P is the total number of positions on the ring.

Properties

- Balance
  - Let $k \in \mathcal{U}$ be a key, chosen uniformly at random.
  - A hash algorithm achieves balance if and only if k has an equal probability of being mapped to each bucket in $\mathcal{W}$.
  - We note that our proposals are orthogonal to the property of *Balance*.
  - Since the resources are mapped to the ring multiple times as per the techniques outlined in Thresh-Hash and Buck-Hash, the mapping is no longer uniform anymore.
  - A key $k \in \mathcal{U}$ has a probability of $\frac{1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil}{P}$ for being mapped to a resource in $\mathcal{S}_{high}$ as per technique Thresh-Hash and a probability of $\frac{\left\lceil \frac{M_s}{M_{min}} \right\rceil}{P}$ for being mapped to a resource as per technique Buck-Hash, where P is the total number of positions on the ring.

Properties

- Balance
  - Let $k \in \mathcal{U}$ be a key, chosen uniformly at random.
  - A hash algorithm achieves balance if and only if k has an equal probability of being mapped to each bucket in $\mathcal{W}$.
  - We note that our proposals are orthogonal to the property of *Balance*.
  - Since the resources are mapped to the ring multiple times as per the techniques outlined in Thresh-Hash and Buck-Hash, the mapping is no longer uniform anymore.
  - A key $k \in \mathcal{U}$ has a probability of $\frac{1 + \left\lceil \frac{w_s m_s - w_{s'} m_{s'}}{w_s} \right\rceil}{P}$ for being mapped to a resource in $\mathcal{S}_{high}$ as per technique Thresh-Hash and a probability of $\frac{\left\lceil \frac{M_s}{M_{min}} \right\rceil}{P}$ for being mapped to a resource as per technique Buck-Hash, where P is the total number of positions on the ring.

Properties

- Balance
  - We also observe that in technique Thresh-Hash, we assign resources to buckets in a greedy manner, where the resource with the highest metric, if available, is paired with the bucket being added to the working set.
  - This also induces a bias towards the buckets that are paired to the resources with higher metric.
  - However, we note here that as $w_s m_s - w_{s'} m_{s'} \longrightarrow 0$, the hashing becomes more uniform in technique Thresh-Hash. Similarly, as $\left\lceil \frac{M_s}{M_{min}} \right\rceil \longrightarrow 1$, the hashing becomes more uniform in technique Buck-Hash.

Properties

- Balance
  - We also observe that in technique Thresh-Hash, we assign resources to buckets in a greedy manner, where the resource with the highest metric, if available, is paired with the bucket being added to the working set.
  - This also induces a bias towards the buckets that are paired to the resources with higher metric.
  - However, we note here that as $w_s m_s - w_{s'} m_{s'} \longrightarrow 0$, the hashing becomes more uniform in technique Thresh-Hash. Similarly, as $\left\lceil \frac{M_s}{M_{min}} \right\rceil \longrightarrow 1$, the hashing becomes more uniform in technique Buck-Hash.

Properties

- Balance
  - We also observe that in technique Thresh-Hash, we assign resources to buckets in a greedy manner, where the resource with the highest metric, if available, is paired with the bucket being added to the working set.
  - This also induces a bias towards the buckets that are paired to the resources with higher metric.
  - However, we note here that as $w_s m_s - w_{s'} m_{s'} \longrightarrow 0$, the hashing becomes more uniform in technique Thresh-Hash. Similarly, as $\left\lceil \frac{M_s}{M_{min}} \right\rceil \longrightarrow 1$, the hashing becomes more uniform in technique Buck-Hash.

Properties

- Balance
  - We also observe that in technique Thresh-Hash, we assign resources to buckets in a greedy manner, where the resource with the highest metric, if available, is paired with the bucket being added to the working set.
  - This also induces a bias towards the buckets that are paired to the resources with higher metric.
  - However, we note here that as $w_s m_s - w_{s'} m_{s'} \longrightarrow 0$, the hashing becomes more uniform in technique Thresh-Hash. Similarly, as $\left\lceil \frac{M_s}{M_{min}} \right\rceil \longrightarrow 1$, the hashing becomes more uniform in technique Buck-Hash.

Properties

- Consistency
  - A hash algorithm is consistent if and only if it achieves both minimal disruption and balance.
  - Since our proposed techniques do not achieve balance, instead they move away from balance to address heterogeneity, the hashing is no longer consistent.
  - We summarise the behaviour of AnchorHash, Thresh-Hash and Buck-Hash in the below table.

Hashing Technique	Minimal Disruption	Balance	Consistency	Addresses Heterogeneity
AnchorHash	✓	✓	✓	✗
Thresh-Hash	✓	✗	✗	✓
Buck-Hash	✓	✗	✗	✓

Properties

- Consistency
  - A hash algorithm is consistent if and only if it achieves both minimal disruption and balance.
  - Since our proposed techniques do not achieve balance, instead they move away from balance to address heterogeneity, the hashing is no longer consistent.
  - We summarise the behaviour of AnchorHash, Thresh-Hash and Buck-Hash in the below table.

Hashing Technique	Minimal Disruption	Balance	Consistency	Addresses Heterogeneity
AnchorHash	✓	✓	✓	✗
Thresh-Hash	✓	✗	✗	✓
Buck-Hash	✓	✗	✗	✓

Properties

- Consistency
  - A hash algorithm is consistent if and only if it achieves both minimal disruption and balance.
  - Since our proposed techniques do not achieve balance, instead they move away from balance to address heterogeneity, the hashing is no longer consistent.
  - We summarise the behaviour of AnchorHash, Thresh-Hash and Buck-Hash in the below table.

Hashing Technique	Minimal Disruption	Balance	Consistency	Addresses Heterogeneity
AnchorHash	✓	✓	✓	✗
Thresh-Hash	✓	✗	✗	✓
Buck-Hash	✓	✗	✗	✓

Properties

- Consistency
  - A hash algorithm is consistent if and only if it achieves both minimal disruption and balance.
  - Since our proposed techniques do not achieve balance, instead they move away from balance to address heterogeneity, the hashing is no longer consistent.
  - We summarise the behaviour of AnchorHash, Thresh-Hash and Buck-Hash in the below table.

Hashing Technique	Minimal Disruption	Balance	Consistency	Addresses Heterogeneity
AnchorHash	✓	✓	✓	✗
Thresh-Hash	✓	✗	✗	✓
Buck-Hash	✓	✗	✗	✓

Properties

- Consistency
  - A hash algorithm is consistent if and only if it achieves both minimal disruption and balance.
  - Since our proposed techniques do not achieve balance, instead they move away from balance to address heterogeneity, the hashing is no longer consistent.
  - We summarise the behaviour of AnchorHash, Thresh-Hash and Buck-Hash in the below table.

Hashing Technique	Minimal Disruption	Balance	Consistency	Addresses Heterogeneity
AnchorHash	✓	✓	✓	✗
Thresh-Hash	✓	✗	✗	✓
Buck-Hash	✓	✗	✗	✓

Experiments

- We simulated[†] the mapping of resources to positions on the ring using the two techniques that we had proposed in Thresh-Hash and Buck-Hash.
- The resources are identified by their indices which start from 0 and end at N.
- The hash function that we had used is as follows:

$$h(i) = ((i\%P) + \eta)\%P$$

where i is the index of the resource, P is number of positions on the ring and $\eta \in \{1, \cdots, P\}$ is a randomly generated offset. We utilise the offset to emulate multiple hash functions.

- We present the plots for Thresh-Hash and Buck-Hash techniques with $P = 300$ and $N = 100$.
- We have assumed that the metric $M \in [0, 1]$.
- For the plot using technique Thresh-Hash, $w_s : w_{s'} = 7 : 3$ and $M_{s'} < 0.6 \ \forall \ s' \in \mathcal{S}_{low}$

[†]The simulations were performed using Python scripts which can be accessed at https://github.com/ksanu1998/Heterogeneity-Aware-Hashing

Experiments

- We simulated[†] the mapping of resources to positions on the ring using the two techniques that we had proposed in Thresh-Hash and Buck-Hash.
- The resources are identified by their indices which start from 0 and end at N.
- The hash function that we had used is as follows:

$$h(i) = ((i\%P) + \eta)\%P$$

where i is the index of the resource, P is number of positions on the ring and $\eta \in \{1, \cdots, P\}$ is a randomly generated offset. We utilise the offset to emulate multiple hash functions.

- We present the plots for Thresh-Hash and Buck-Hash techniques with $P = 300$ and $N = 100$.
- We have assumed that the metric $M \in [0, 1]$.
- For the plot using technique Thresh-Hash, $w_s : w_{s'} = 7 : 3$ and $M_{s'} < 0.6 \ \forall \ s' \in \mathcal{S}_{low}$

[†]The simulations were performed using Python scripts which can be accessed at https://github.com/ksanu1998/Heterogeneity-Aware-Hashing

Experiments

- We simulated[†] the mapping of resources to positions on the ring using the two techniques that we had proposed in Thresh-Hash and Buck-Hash.
- The resources are identified by their indices which start from 0 and end at N.
- The hash function that we had used is as follows:

$$h(i) = ((i\%P) + \eta)\%P$$

where i is the index of the resource, P is number of positions on the ring and $\eta \in \{1, \cdots, P\}$ is a randomly generated offset. We utilise the offset to emulate multiple hash functions.

- We present the plots for Thresh-Hash and Buck-Hash techniques with $P = 300$ and $N = 100$.
- We have assumed that the metric $M \in [0, 1]$.
- For the plot using technique Thresh-Hash, $w_s : w_{s'} = 7 : 3$ and $M_{s'} < 0.6 \ \forall \ s' \in \mathcal{S}_{low}$

[†]The simulations were performed using Python scripts which can be accessed at https://github.com/ksanu1998/Heterogeneity-Aware-Hashing

Experiments

- We simulated[†] the mapping of resources to positions on the ring using the two techniques that we had proposed in Thresh-Hash and Buck-Hash.
- The resources are identified by their indices which start from 0 and end at N.
- The hash function that we had used is as follows:

$$h(i) = ((i\%P) + \eta)\%P$$

where i is the index of the resource, P is number of positions on the ring and $\eta \in \{1, \cdots, P\}$ is a randomly generated offset. We utilise the offset to emulate multiple hash functions.

- We present the plots for Thresh-Hash and Buck-Hash techniques with $P = 300$ and $N = 100$.
- We have assumed that the metric $M \in [0, 1]$.
- For the plot using technique Thresh-Hash, $w_s : w_{s'} = 7 : 3$ and $M_{s'} < 0.6 \ \forall \ s' \in \mathcal{S}_{low}$

[†]The simulations were performed using Python scripts which can be accessed at https://github.com/ksanu1998/Heterogeneity-Aware-Hashing

Experiments

- We simulated[†] the mapping of resources to positions on the ring using the two techniques that we had proposed in Thresh-Hash and Buck-Hash.
- The resources are identified by their indices which start from 0 and end at N.
- The hash function that we had used is as follows:

$$h(i) = ((i\%P) + \eta)\%P$$

where i is the index of the resource, P is number of positions on the ring and $\eta \in \{1, \cdots, P\}$ is a randomly generated offset. We utilise the offset to emulate multiple hash functions.

- We present the plots for Thresh-Hash and Buck-Hash techniques with $P = 300$ and $N = 100$.
- We have assumed that the metric $M \in [0, 1]$.
- For the plot using technique Thresh-Hash, $w_s : w_{s'} = 7 : 3$ and $M_{s'} < 0.6 \ \forall \ s' \in \mathcal{S}_{low}$

[†]The simulations were performed using Python scripts which can be accessed at https://github.com/ksanu1998/Heterogeneity-Aware-Hashing

Experiments

- We simulated[†] the mapping of resources to positions on the ring using the two techniques that we had proposed in Thresh-Hash and Buck-Hash.
- The resources are identified by their indices which start from 0 and end at N.
- The hash function that we had used is as follows:

$$h(i) = ((i\%P) + \eta)\%P$$

where i is the index of the resource, P is number of positions on the ring and $\eta \in \{1, \cdots, P\}$ is a randomly generated offset. We utilise the offset to emulate multiple hash functions.

- We present the plots for Thresh-Hash and Buck-Hash techniques with $P = 300$ and $N = 100$.
- We have assumed that the metric $M \in [0, 1]$.
- For the plot using technique Thresh-Hash, $w_s : w_{s'} = 7 : 3$ and $M_{s'} < 0.6 \ \forall \ s' \in \mathcal{S}_{low}$

[†]The simulations were performed using Python scripts which can be accessed at https://github.com/ksanu1998/Heterogeneity-Aware-Hashing

Experiments

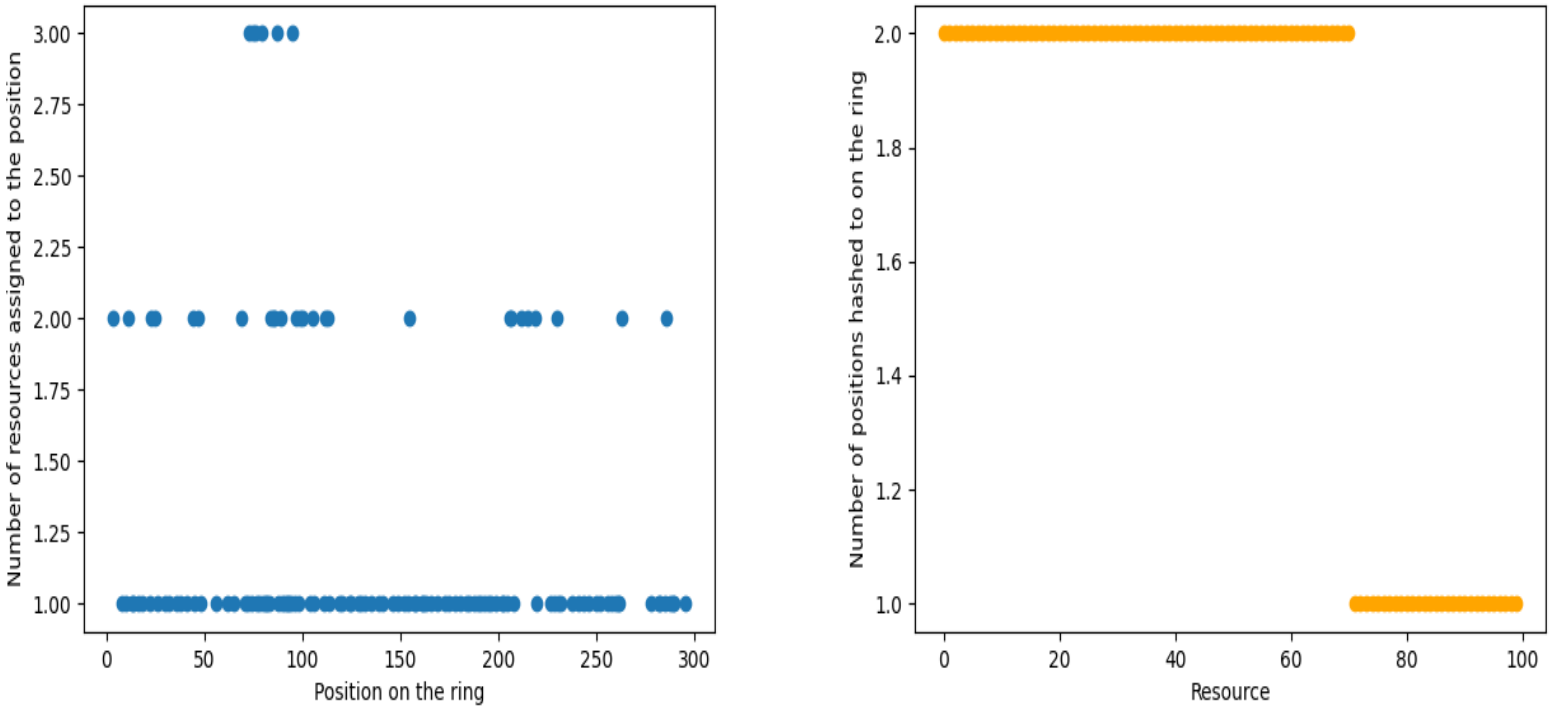

Figure: Technique Thresh-Hash

We observe that most of the positions on the ring are assigned 1 resource, some of the positions have been assigned 2 resources and very few have been assigned 3 resources. All the resources with a higher metric have been mapped to two positions on the ring.

Experiments

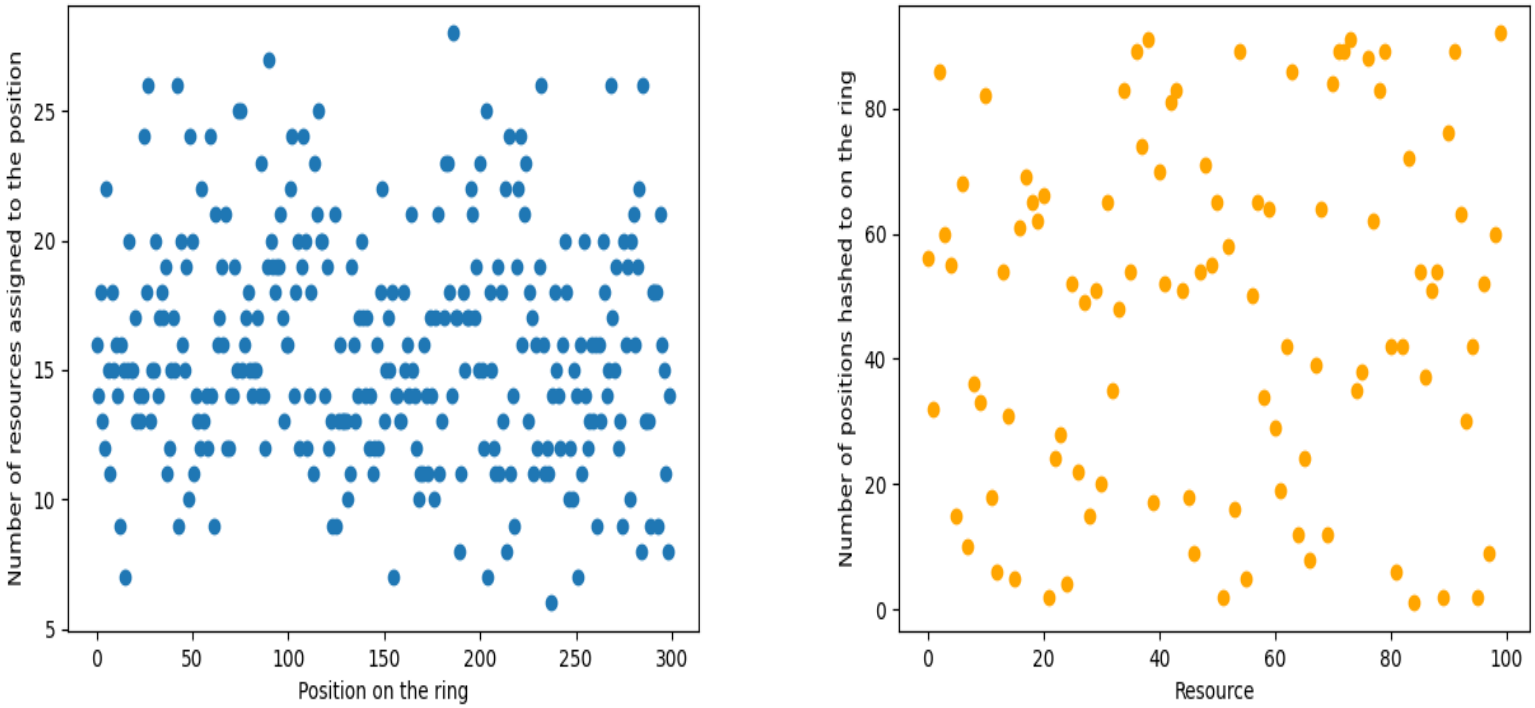

Figure: Technique Buck-Hash

It is observed that a wide range of resources are allocated to different positions on the ring. Also, resources having higher metric are mapped to multiple positions on the ring.

Experiments

- In both cases, we would like highlight the fact that the same resource may be potentially hashed to multiple positions on the ring and the hashing function does not discriminate between resources.
- Ultimately, it is this multiple hashing that addresses the justified distribution of load in both the techniques. We also observed that in technique Buck-Hash, as $P - N$ increases, the mapping of resources to positions on the ring resembles the one obtained using technique Thresh-Hash.
- We suggest that one uses technique Buck-Hash if the resource metrics do not vary considerably and technique Thresh-Hash otherwise. To highlight the intuition behind this suggestion of ours, consider the figures for Buck-Hash in the next slide.

Experiments

- In both cases, we would like highlight the fact that the same resource may be potentially hashed to multiple positions on the ring and the hashing function does not discriminate between resources.
- Ultimately, it is this multiple hashing that addresses the justified distribution of load in both the techniques. We also observed that in technique Buck-Hash, as $P - N$ increases, the mapping of resources to positions on the ring resembles the one obtained using technique Thresh-Hash.
- We suggest that one uses technique Buck-Hash if the resource metrics do not vary considerably and technique Thresh-Hash otherwise. To highlight the intuition behind this suggestion of ours, consider the figures for Buck-Hash in the next slide.

Experiments

- In both cases, we would like highlight the fact that the same resource may be potentially hashed to multiple positions on the ring and the hashing function does not discriminate between resources.
- Ultimately, it is this multiple hashing that addresses the justified distribution of load in both the techniques. We also observed that in technique Buck-Hash, as $P - N$ increases, the mapping of resources to positions on the ring resembles the one obtained using technique Thresh-Hash.
- We suggest that one uses technique Buck-Hash if the resource metrics do not vary considerably and technique Thresh-Hash otherwise. To highlight the intuition behind this suggestion of ours, consider the figures for Buck-Hash in the next slide.

Experiments

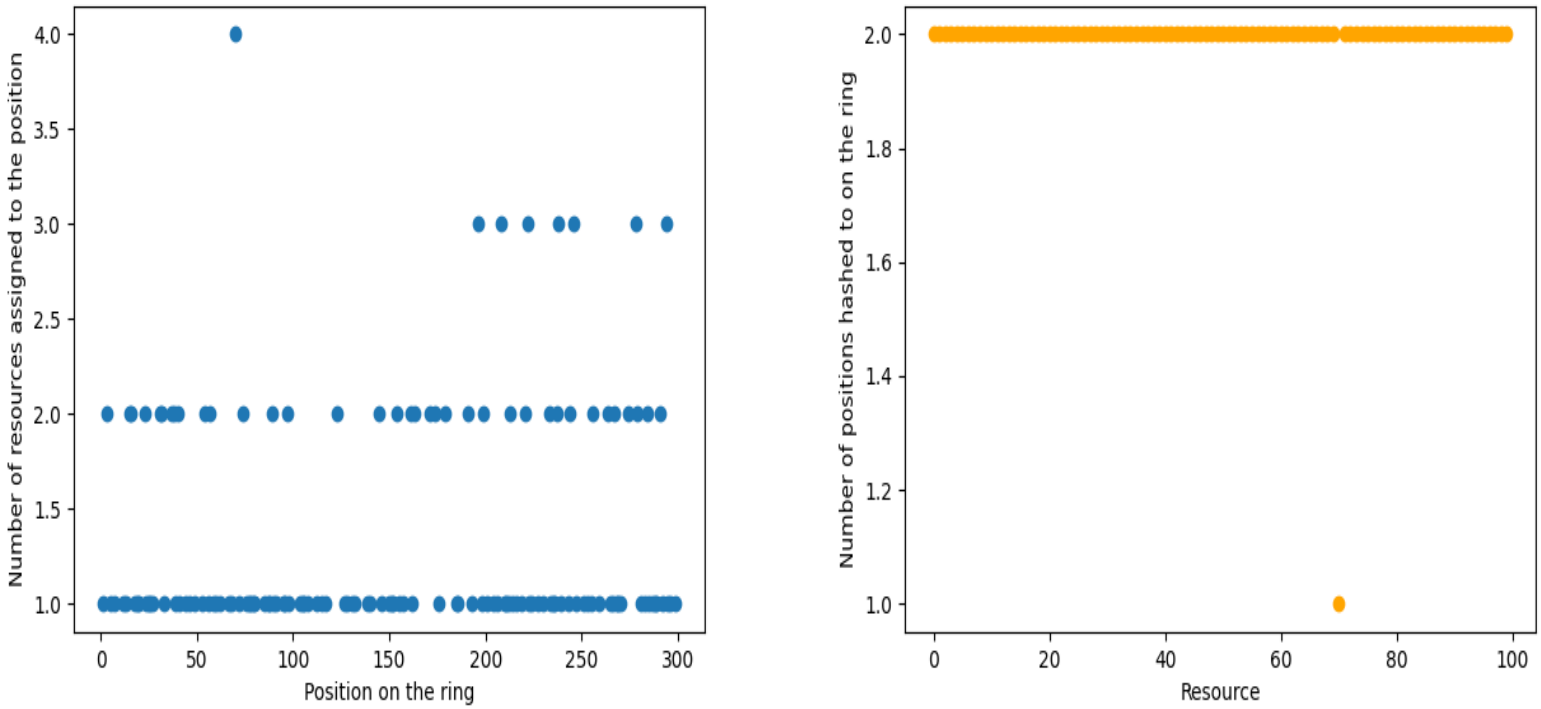

Figure: Technique Buck-Hash with metric values constricted to a narrow range

Buck-Hash with $P = 300$, $N = 100$ and $M_s \in [0.8, 1]$. It is observed here that the seemingly random nature of mapping in the earlier case has now become more organised and also the maximum number of positions on the ring to which a resource is mapped has now reduced to 2.

Conclusion

- We reviewed the literature on CH and appreciated its widespread use in various distributed applications.
- We presented the crux of the working of AnchorHash and discussed the allocation of keys to resources when capacities are varying in nature.
- We then proposed some heterogeneity-aware hashing techniques which can be used along with AnchorHash framework.
- We analysed the proposed techniques in light of some CH properties and then simulated the mapping of resources to the positions on the ring based on the proposed techniques.
- We finally presented our insights based on the simulations.

Conclusion

- We reviewed the literature on CH and appreciated its widespread use in various distributed applications.
- We presented the crux of the working of AnchorHash and discussed the allocation of keys to resources when capacities are varying in nature.
- We then proposed some heterogeneity-aware hashing techniques which can be used along with AnchorHash framework.
- We analysed the proposed techniques in light of some CH properties and then simulated the mapping of resources to the positions on the ring based on the proposed techniques.
- We finally presented our insights based on the simulations.

Conclusion

- We reviewed the literature on CH and appreciated its widespread use in various distributed applications.
- We presented the crux of the working of AnchorHash and discussed the allocation of keys to resources when capacities are varying in nature.
- We then proposed some heterogeneity-aware hashing techniques which can be used along with AnchorHash framework.
- We analysed the proposed techniques in light of some CH properties and then simulated the mapping of resources to the positions on the ring based on the proposed techniques.
- We finally presented our insights based on the simulations.

Conclusion

- We reviewed the literature on CH and appreciated its widespread use in various distributed applications.
- We presented the crux of the working of AnchorHash and discussed the allocation of keys to resources when capacities are varying in nature.
- We then proposed some heterogeneity-aware hashing techniques which can be used along with AnchorHash framework.
- We analysed the proposed techniques in light of some CH properties and then simulated the mapping of resources to the positions on the ring based on the proposed techniques.
- We finally presented our insights based on the simulations.

Conclusion

- We reviewed the literature on CH and appreciated its widespread use in various distributed applications.
- We presented the crux of the working of AnchorHash and discussed the allocation of keys to resources when capacities are varying in nature.
- We then proposed some heterogeneity-aware hashing techniques which can be used along with AnchorHash framework.
- We analysed the proposed techniques in light of some CH properties and then simulated the mapping of resources to the positions on the ring based on the proposed techniques.
- We finally presented our insights based on the simulations.

Future Work

- A full-scale test can be done for a more rigorous analysis of the feasibility and scalability of the proposed techniques.
- Also, the problem of multiple resources being hashed to the same position on the ring needs to be addressed.
- We may even change the nature of mapping between buckets and resources to accommodate consistency to a greater extant.

Future Work

- A full-scale test can be done for a more rigorous analysis of the feasibility and scalability of the proposed techniques.
- Also, the problem of multiple resources being hashed to the same position on the ring needs to be addressed.
- We may even change the nature of mapping between buckets and resources to accommodate consistency to a greater extant.

Future Work

- A full-scale test can be done for a more rigorous analysis of the feasibility and scalability of the proposed techniques.
- Also, the problem of multiple resources being hashed to the same position on the ring needs to be addressed.
- We may even change the nature of mapping between buckets and resources to accommodate consistency to a greater extant.

References

[1] Giuseppe DeCandia et al. "Dynamo: amazon's highly available key-value store". In: *ACM SIGOPS operating systems review* 41.6 (2007), pp. 205–220.

[2] John Dilley et al. "Globally distributed content delivery". In: *IEEE Internet Computing* 6.5 (2002), pp. 50–58.

[3] Daniel E Eisenbud et al. "Maglev: A fast and reliable software network load balancer". In: *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*. 2016, pp. 523–535.

[4] David Karger et al. "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web". In: *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. 1997, pp. 654–663.

[5] Avinash Lakshman and Prashant Malik. "Cassandra: a decentralized structured storage system". In: *ACM SIGOPS Operating Systems Review* 44.2 (2010), pp. 35–40.

[6] Gal Mendelson et al. "AnchorHash: A Scalable Consistent Hash". In: *arXiv preprint arXiv:1812.09674* (2018).

References (cont.)

[7] KyoungSoo Park and Vivek S Pai. "Scale and Performance in the CoBlitz Large-File Distribution Service.". In: *NSDI*. 2006.

[8] Ion Stoica et al. "Chord: a scalable peer-to-peer lookup protocol for internet applications". In: *IEEE/ACM Transactions on networking* 11.1 (2003), pp. 17–32.

[9] David G Thaler and Chinya V Ravishankar. "Using name-based mappings to increase hit rates". In: *IEEE/ACM Transactions on networking* 6.1 (1998), pp. 1–14.

Thank you!