



**USC Viterbi**  
School of Engineering

# **D-Zipfian: A Decentralized Implementation of Zipfian**

Sumita Barahmand and Shahram Ghandeharizadeh  
Database Lab, University of Southern California  
{barahman, shahram}@usc.edu

June 2013

- Benchmarking
  - Modeling Applications
    - Zipfian distribution
- Scalable Benchmarks
  - A current limitation
  - Solutions
    - Replicated Zipfian
    - Crude
    - Decentralized Zipfian

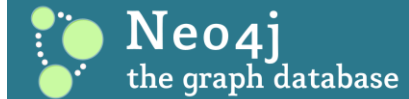
- Explosion in the number of data stores developed for OLTP and social networking applications.

— SQL, NoSQL, NewSQL, Graph databases and etc.



ORACLE®

VoltDB



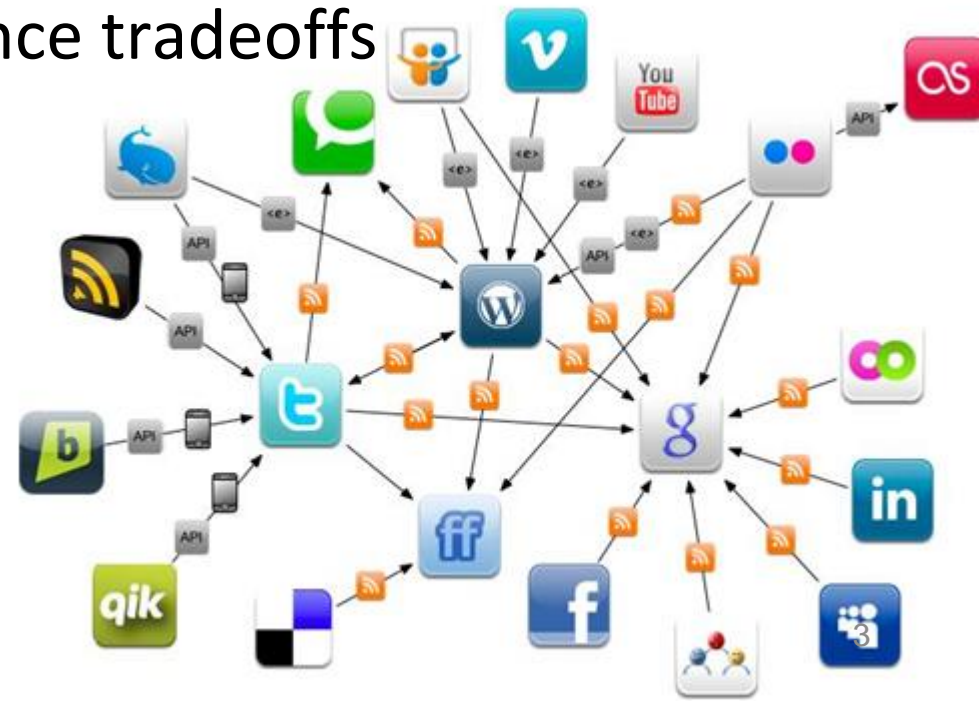
- Benchmarks developed to evaluate, test and understand the performance tradeoffs between data stores for different applications.

— TPC-C

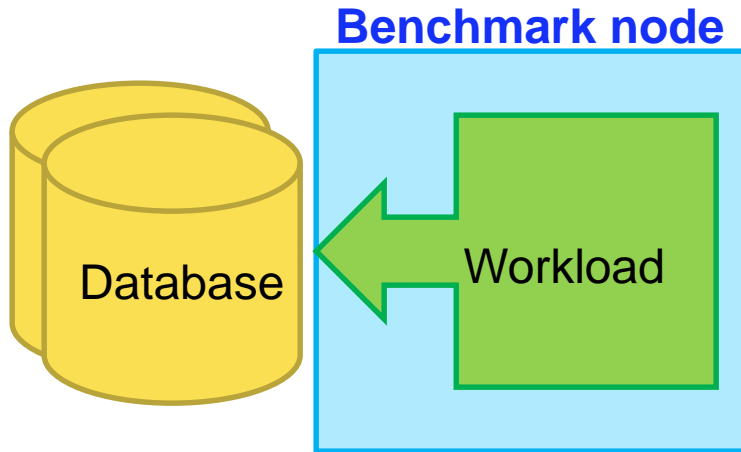
— YCSB/YCSB++

— BG

— LinkBench



- Database benchmarks mimic a particular kind of application workload on the database system.
- Benchmark objective: Evaluate and test database systems accurately.
- An accurate benchmark:
  - Models the application accurately.
  - Gathers accurate data.
  - Produces results which are reproducible and repeatable.
  - Produces meaningful results which are not misinterpreted.



**WHAT?** What actions to issue?  
What data items to reference?

## TPC-C:

**5 Actions:** Entering and delivering orders, recording payments, checking order status and monitoring warehouse inventory.

**Data items:** customers and items.

## YCSB/YCSB++:

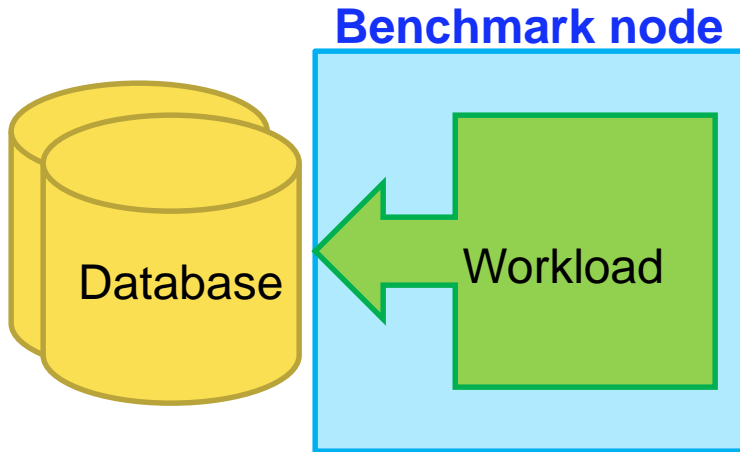
**5 Actions:** Read, insert, update, delete, scan.

**Data items:** records.

## BG:

**11 Actions:** ViewProfile, ListFriends, InviteFriends, ViewTopKResources, etc.

**Data items:** users, resources and manipulations.



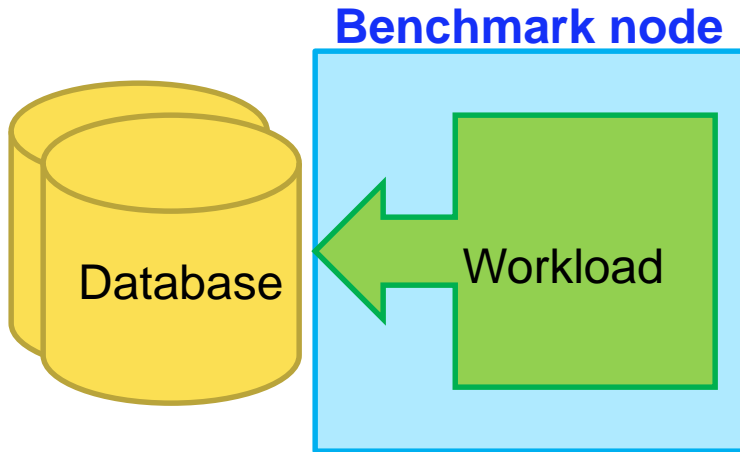
## WHAT?

What actions to issue?  
What data items to reference?

## WHEN?

When to issue the actions  
against the database?

- Closed simulation model
- Open simulation model



**WHAT?** What actions to issue?  
What data items to reference?

- **Expected distribution:**
  - Expected probability of reference for each data item.
  - It is given as an input to the benchmark and is application specific.
- **Observed distribution:**
  - Probability of reference for each data item computed after the benchmark is executed.
  - This value is computed by dividing the number of requests for a data item by the total number of requests issued for all items.
- **Chi square analysis:**
  - Allows us to compare a collection of observed distribution with a theoretical expected distribution.

$$\chi^2 = \sum \frac{(\text{expected probability} - \text{observed probability})^2}{\text{expected probability}}$$





- Random distribution of access is not realistic due to Zipf's law.
- This law states that given some collection of data items, the frequency of any data item is inversely proportional to its rank in its frequency table.
- Zipfian distribution is characterized by an exponent,  $\Theta$ .



$$p_i(M, \theta) = \frac{1}{i^{(1-\theta)} \sum_{m=1}^M \left( \frac{1}{m^{(1-\theta)}} \right)}$$

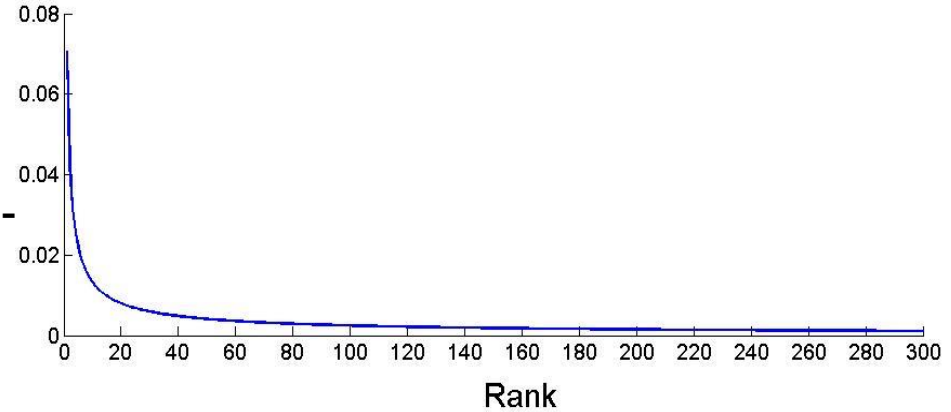
- 80 - 20 Rule:

80% of requests (ticket sales, frequency of words , profile look-ups) reference  
20% of data items (movies opening on a weekend, words uttered in natural  
language, members of a social networking site).

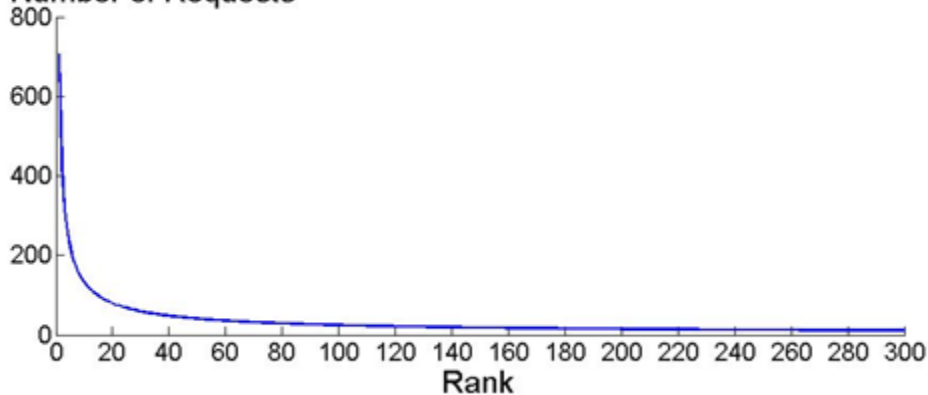
# Zipfian Distribution

- $M=300$  items.
- $\Theta = 0.27$ .
- Total number of requests = 10,000.
- A few items have a high probability of reference.
- A medium number of items have a middle-of-the-road probability of reference.
- A huge number of items have a very low probability of reference.

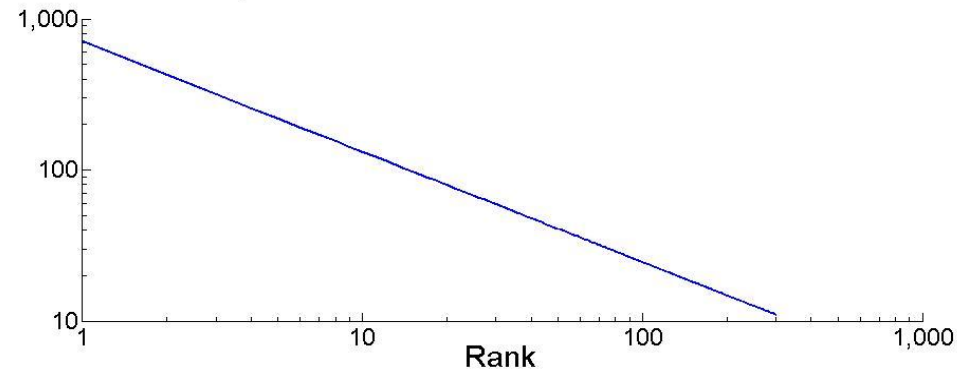
Probability of Reference



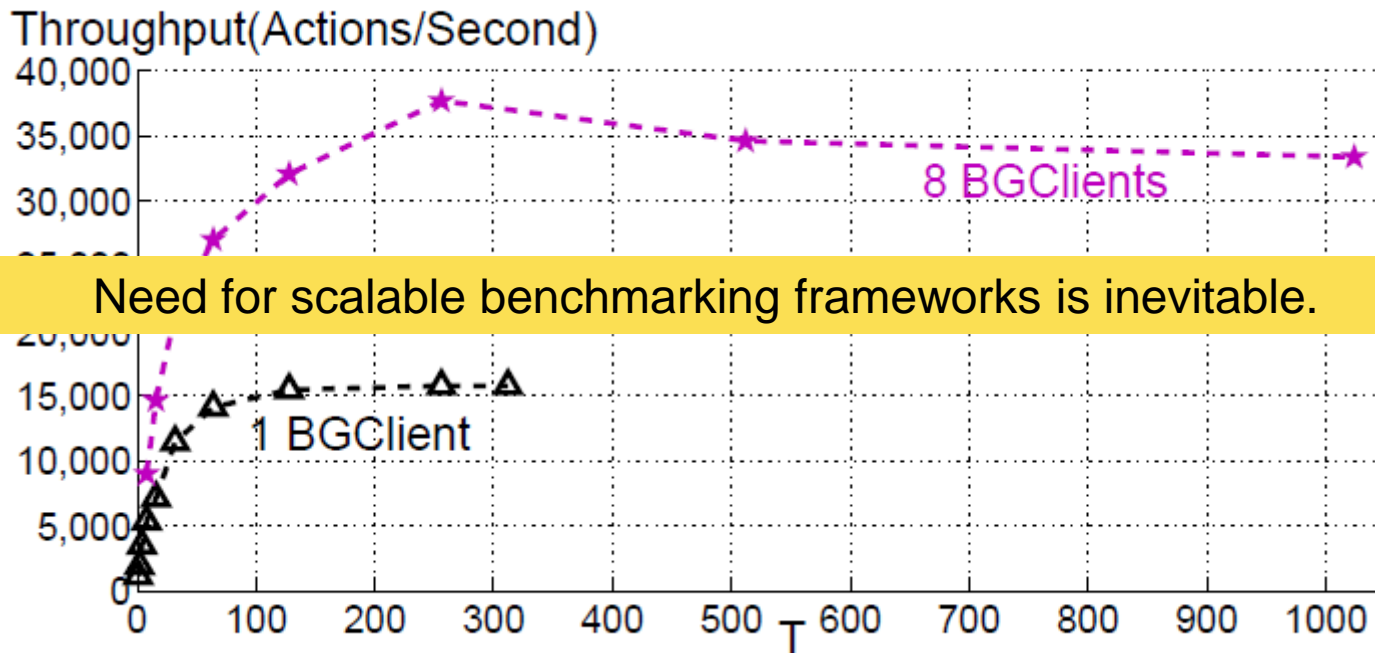
Number of Requests



Number of Requests



- Assumption: Rate the throughput of a database under heavy load or strict service level agreement requirements.
- Today's data stores process requests at such a high rate that one *benchmark node* may not be sufficient to rate them accurately.
  - One node may use its resources fully and fail to generate work at a sufficiently high rate to evaluate its target data store.
- To address this challenge, a benchmarking framework should utilize multiple nodes to generate work for its target data store.



- BG social benchmark's ViewProfile workload with 10,000 members.
- Every BGClient is a single benchmarking node, issuing requests to the data store independently.

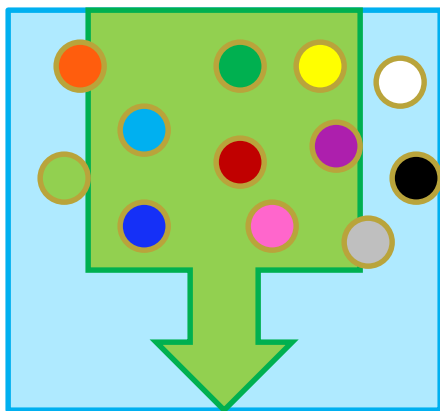
- How do multiple nodes produce requests such that their overall observed distribution conforms to a pre-specified Zipfian distribution?
  - Requests generated by multiple nodes should resemble a Zipfian distribution.
  - Probability of referencing data items should be independent of the degree of parallelism, i.e., number of employed nodes.
  - The distribution generated by the nodes should be independent of the performance of the nodes (rate at which they generate requests).

- Replication: Replicated-Zipfian (R-Zipfian)
  - Each node accesses the entire population.
  - Each node issues request based on a Zipfian distribution.
- Partitioning: Decentralized-Zipfian (D-Zipfian)
  - Each node accesses a unique fraction of the entire population.
  - Each node issues requests based on a Zipfian distribution.

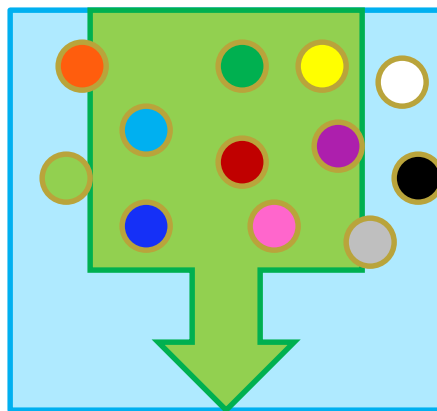
- Replication: R-Zipfian
  - Each node accesses the entire population.
  - Each node issues request based on a Zipfian distribution.
- Partitioning: D-Zipfian
  - Each node accesses a unique fraction of the entire population.
  - Each node issues requests based on a Zipfian distribution.
- **Contribution:**
  - **D-Zipfian**
    - **Scalable benchmarking framework: Uses additional nodes without incurring additional overhead.**
    - **Workloads consisting of a mix of read and write actions.**
    - **Workloads where benchmarking nodes must reference unique data items at any instance in time.**



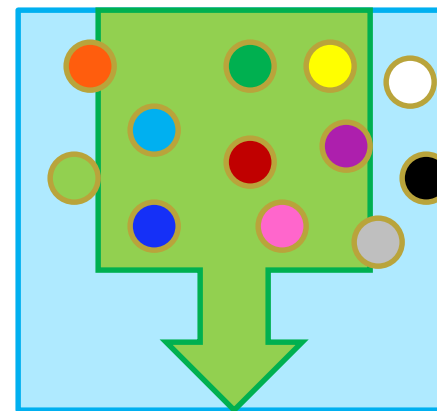
- Requires each node to employ the specified Zipfian distribution with the entire population independently.



Node 1  
M=12 items  
 $\Theta=0.27$   
O=1000  
 $P_1(12,0.27)=0.32$



Node 2  
M=12 items  
 $\Theta=0.27$   
O=1000  
 $P_1(12,0.27)=0.32$



Node 3  
M=12 items  
 $\Theta=0.27$   
O=1000  
 $P_1(12,0.27)=0.32$

**Overall  $P_1 = [(0.32 \times 1000) + (0.32 \times 1000) + (0.32 \times 1000)] / (1000+1000+1000) = 0.32$**



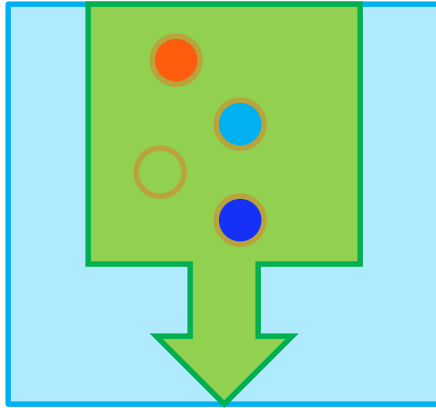
- Requires each node to employ the specified Zipfian distribution with the entire population independently.
- Advantage:
  - Overall of probability of reference for every item remains constant.
  - Distribution is independent of the degree of parallelism.
  - Accommodates heterogeneous nodes where each node produces requests at a different rate.
- Disadvantage:
  - Additional complexity
    - Does not work with workloads that require uniqueness of referenced data items.
    - Depending on the workload the nodes may need to communicate with one another.

- YCSB:
  - With a relational database two nodes may try to insert the same data item (with the same primary key) resulting in integrity constraint violations instead of the intended actions.
- BG:
  - BG measures the amount of unpredictable data produced by a data store using time stamps.
  - R-Zipfian would require BG to utilize synchronized clocks to timestamp the actions else the unpredictable data will not be computed accurately.

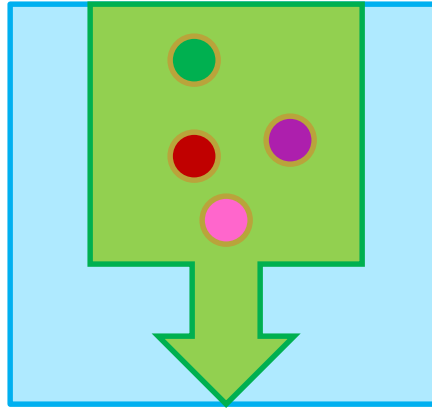
# Naïve Technique – Crude

- Range partition data items across the benchmarking nodes where each node employs the same Zipfian distribution to generate requests.

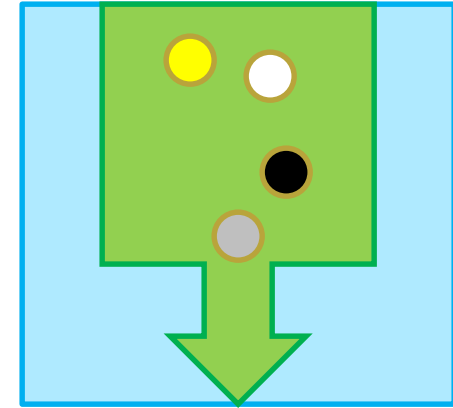
Crude:



Node 1  
M=4 items  
 $\Theta=0.27$   
O=1000  
 $P_1(4,0.27)=0.48,$   
 $P_5=0, P_9=0$



Node 2  
M=4 items  
 $\Theta=0.27$   
O=1000  
 $P_1=0, P_5(4,0.27)=0.48,$   
 $P_9=0$

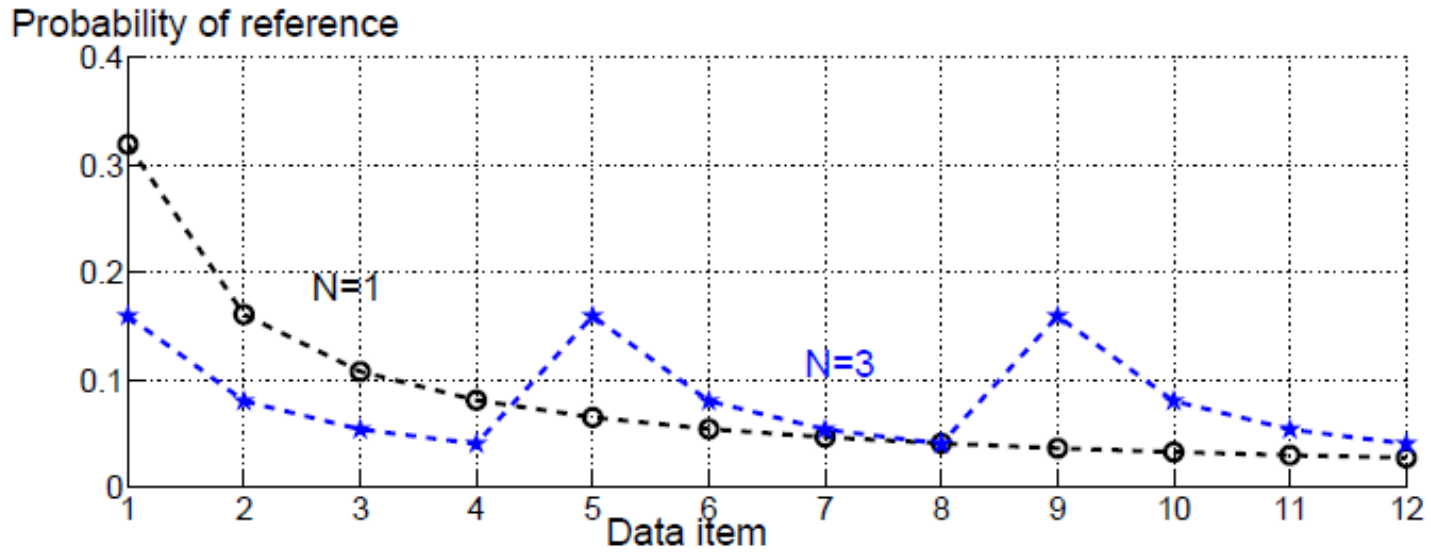


Node 3  
M=4 items  
 $\Theta=0.27$   
O=1000  
 $P_1=0, P_5=0,$   
 $P_9(4,0.27)=0.48$

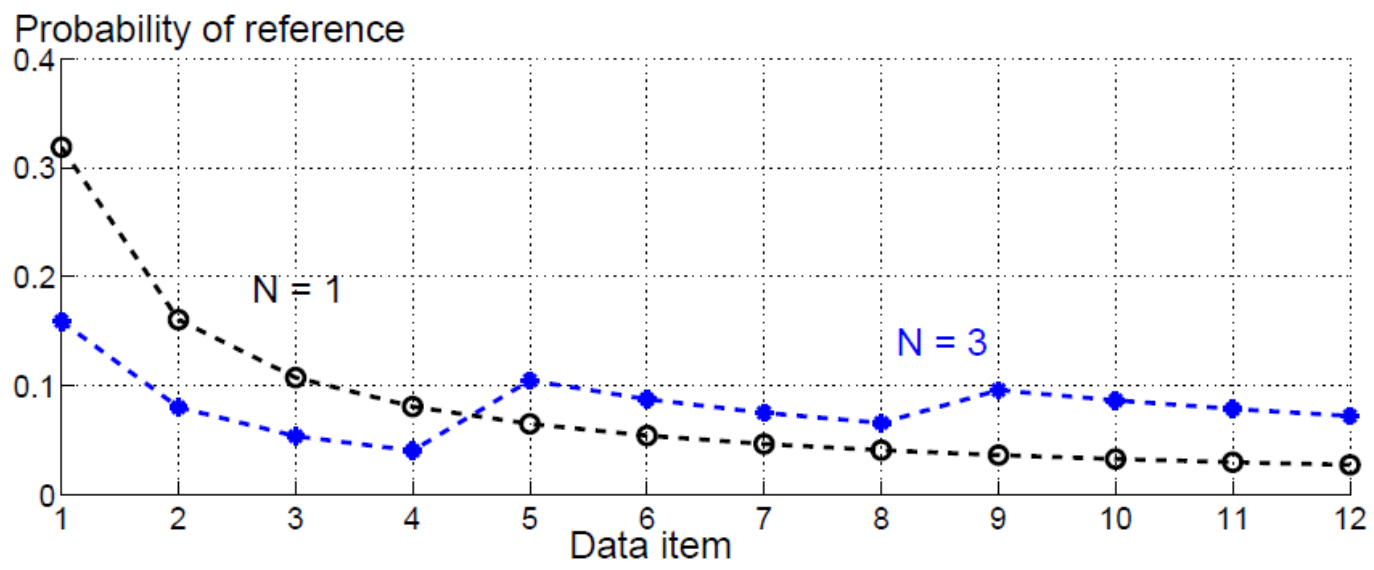
**Overall  $P_1 = [(0.48 \times 1000) + (0 \times 1000) + (0 \times 1000)] / (1000 + 1000 + 1000) = 0.16$**



Crude:



Normalized Crude:



- D-Zipfian employs multiple nodes that reference data items independently.
- Similarity with Crude and Normalized Crude:
  - Database is divided into logical independent fragments where each fragment is assigned to a node.
- Difference with Crude and Normalized Crude:
  - Fragments are created based on a heuristic in an intelligent manner.

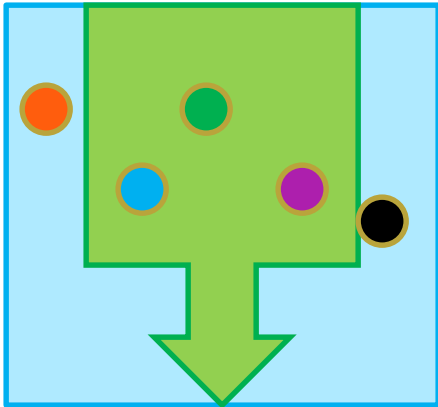
- Computes the probability of referencing each data item considering the entire population using the initial Zipfian distribution characterized by  $\Theta$ .

$$p_i(M, \theta) = \frac{\frac{1}{i^{(1-\theta)}}}{\sum_{m=1}^M \left( \frac{1}{m^{(1-\theta)}} \right)}$$

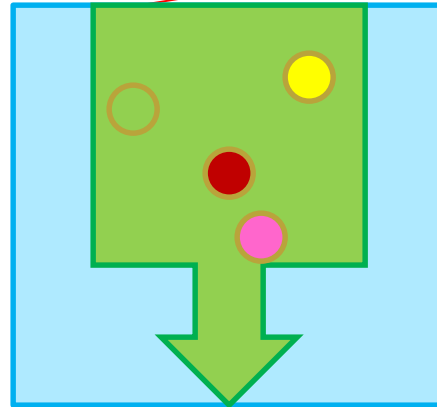
- With N nodes, constructs N fragments such that the sum of the probability of the items assigned to all fragments are equal.

- Assigns each fragment to a node.
- Every node normalizes the probabilities for its assigned items using :

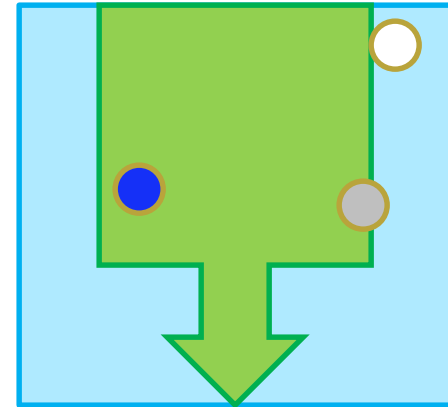
$$p_i = \frac{p_i(M, \theta)}{\sum_{m=1}^{m_k} p_i(M, \theta)} \quad 1/N$$



Node 1  
M=5 items.  
 $\Theta=0.27$   
O=1000  
P1=0



Node 2  
M=4 items.  
 $\Theta=0.27$   
O=1000  
P1=0



Node 3  
M=3 items.  
 $\Theta=0.27$   
O=1000 **0.32**  
 $P1 = P1(12, 0.27) / 0.33$   
 $= 0.97$

Overall P1 =  $[(0 \times 1000) + (0 \times 1000) + (0.97 \times 1000)] / (1000 + 1000 + 1000) = 0.32$

# Example – $M=12$ , $\Theta=0.58$ , $N=3$



Node	Item	Original Probability	Normalized Local Probability	Overall Probability
Node 1	2	0.10731254073162655	0.345522	0.114022
	5	0.06469915081178942	0.208316	0.068744
	7	0.052443697392845726	0.168857	0.055723
	9	0.04456039103539063	0.143474	0.047346
	10	0.04156543530505756	0.133831	0.044164
Sum		<b>0.310581</b>		
Node 2	1	0.1442769977234511	0.445131	0.146893
	4	0.07390960650956815	0.22803	0.07525
	6	0.057813255317337574	0.178368	0.058862
	8	0.048122918873735814	0.148471	0.048996
Sum		<b>0.324123</b>		
Node 3	0	0.2393034684469674	0.655095	0.216181
	3	0.08698516660532968	0.238122	0.07858
	11	0.03900737124690036	0.106783	0.035238
Sum		<b>0.365296</b>		



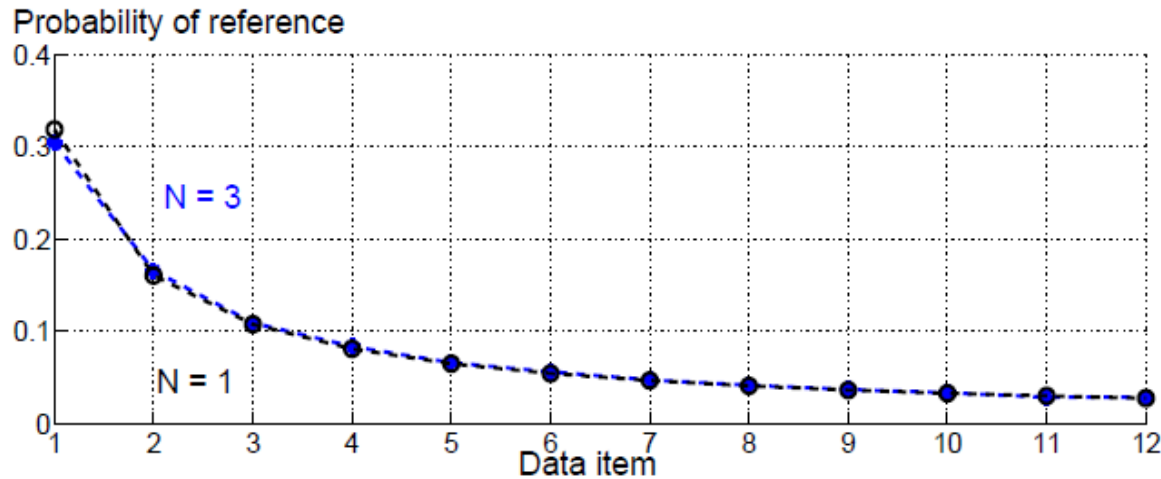
# Example – $M=12$ , $\Theta=0.58$ , $N=3$



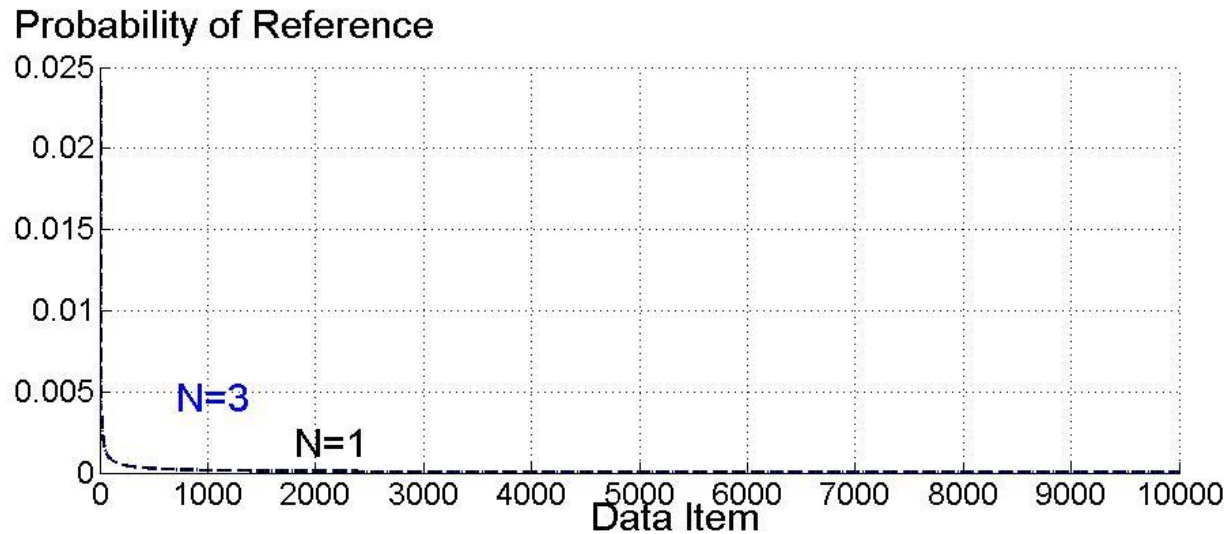
Node	Item	Original Probability	Normalized Local Probability	Overall Probability
Node 1	2	0.10731254073162655	0.345522	0.114022
	5	0.06469915081178942	0.208316	0.068744
	7	0.052443697392845726	0.168857	0.055723
	9	0.04456039103539063	0.143474	0.047346
	10	0.04156543530505756	0.133831	0.044164
Sum		0.310581		
Node 2	1	0.1442769977234511	0.445131	0.146893
	4	0.07390960650956815	0.22803	0.07525
	6	0.057813255317337574	0.178368	0.058862
	8	0.048122918873735814	0.148471	0.048996
Sum		0.324123		
Node 3	0	0.2393034684469674	0.655095	0.216181
	3	0.08698516660532968	0.238122	0.07858
	11	0.03900737124690036	0.106783	0.035238
Sum		0.365296		

- D-Zipfian with homogenous nodes

M=12

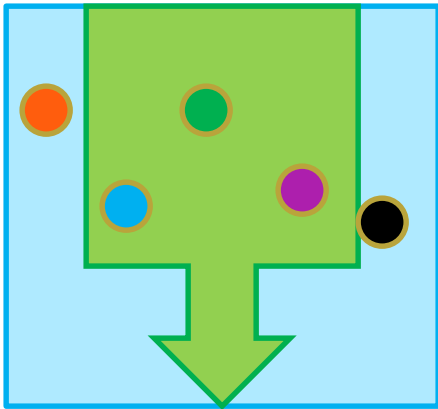


M=10,000

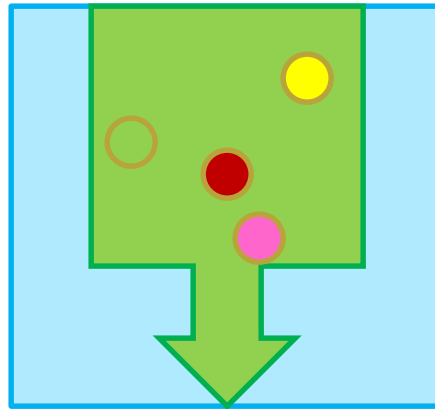


# Heterogeneous Benchmark Nodes

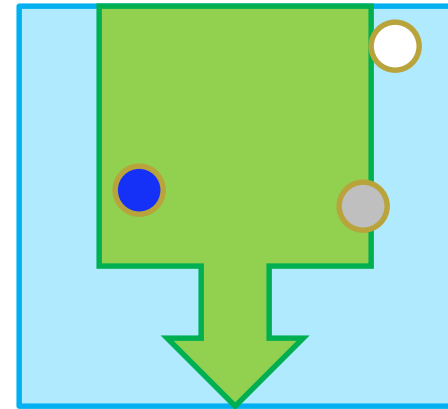
- It is rare to purchase machines that provide same performance.
- Heterogeneous nodes issue requests at different rates.



Node 1  
M=5 items  
 $\Theta=0.27$   
R=2  
O=2000  
P1=0



Node 2  
M=4 items  
 $\Theta=0.27$   
R=1  
O=1000  
P1=0



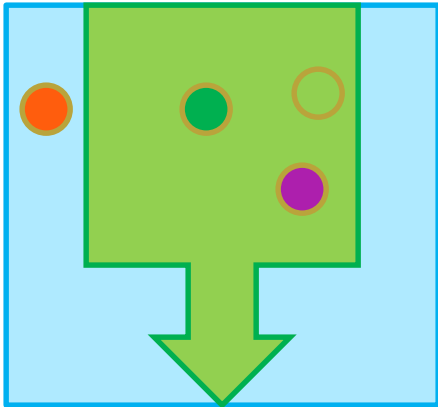
Node 3  
M=3 items  
 $\Theta=0.27$   
R=3  
O=3000  
P1=~~P1(12,0.27)~~/0.33  
=0.97

**0.32**

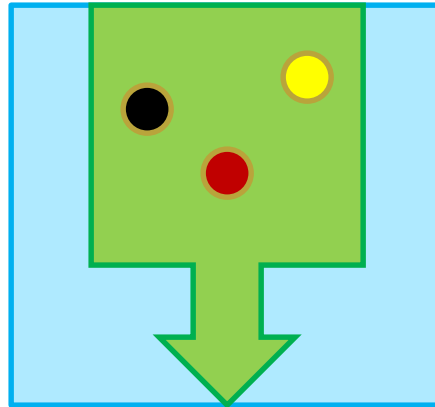
$$\text{Overall P1} = [(0 \times 2000) + (0 \times 1000) + (0.97 \times 3000)] / (2000 + 1000 + 3000) = 0.49$$

# Heterogeneous Benchmark Nodes

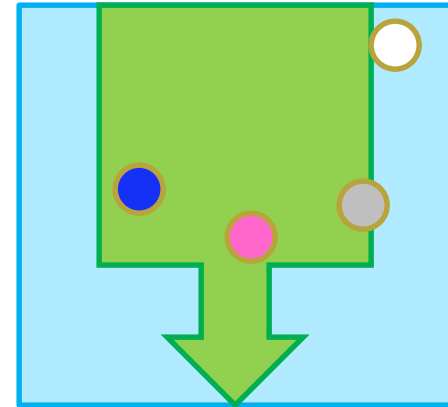
- It is rare to purchase machines that provide same performance.
- Heterogeneous nodes issue requests at different rates.
- Construct fragments for each node such that their total assigned probability is proportional to the rate at which they can issue requests.



Node 1  
R=2  
O=2000  
P1=0



Node 2  
R=1  
O=1000  
P1=0



Node 3  
R=3  
O=3000  
 $P1 = P1(12, 0.27) / (3/6)$   
=0.64

$$\text{Overall } P1 = [(0 \times 2000) + (0 \times 1000) + (0.64 \times 3000)] / (2000 + 1000 + 3000) = 0.32$$

- D-Zipfian is a parallel algorithm that executes on  $N$  nodes.
- D-Zipfian produces a Zipfian distribution that is independent of its degree of parallelism.
- D-Zipfian considers heterogeneity of participating nodes and the rate at which they produce requests in order to produce a distribution comparable to one node generating the distribution.
- D-Zipfian is decentralized and scales to a large number of nodes. It is an essential component of a scalable benchmarking frameworks.

# Questions?

<http://BGBenchmark.org/>

