# Predictive caching in computer grids

## Predictive caching

- Research carried out during 2012
- Aims to increase the performance of hardware and software solutions which store large datasets in a distributed fashion
- Improve performance compared to solutions which do not use predictive caching

## Caching algorithms

### Caching aims

- Improve performance by using fast temporary storage
- Minimize costs by reducing server load and faster data access
- Caching works by storing data onto a fast medium so that future requests can be served quickly
- Cached data can be previous requests or original requests (prefetching)

### Traditional caching algorithms

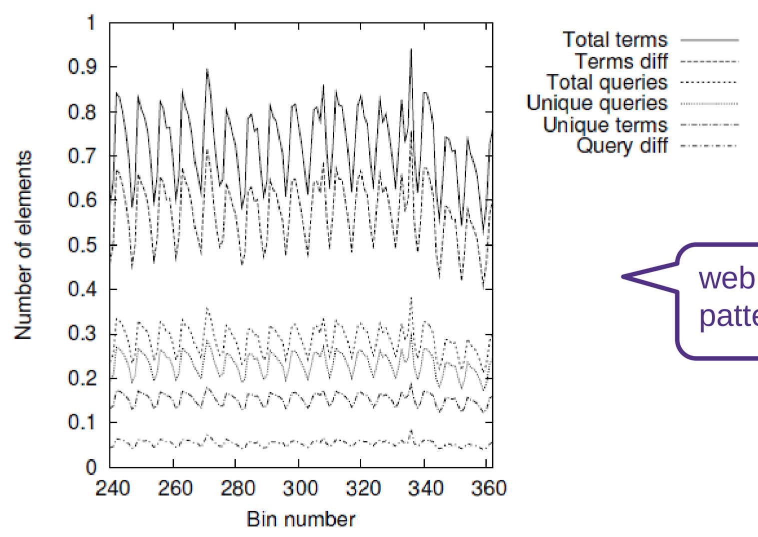
- Least recently used (LRU)
- Least frequently used (LFU)

### Problem

- Traditional algorithms do not work well in a cloud / grid environment
- Significant data transfer costs for shared 'cloud' cache

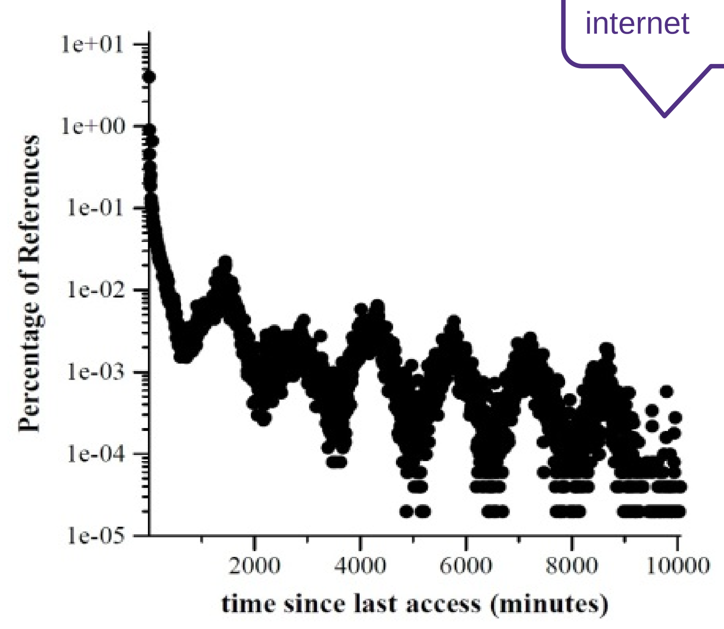
## Predicting the user data access patterns

- Predicting future data requests is key to good caching algorithms
- Real data is often accessed in predictable patterns




web server access pattern (bin = 1 day)

user accessing the internet




- Data access patterns are implemented in the model via calculated object request probabilities

## Innovation

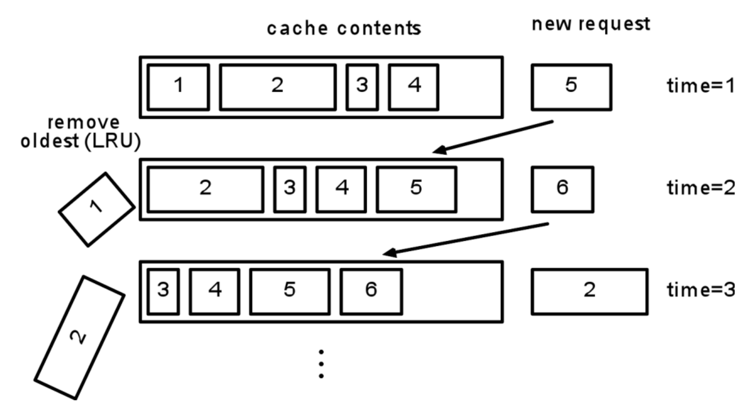
- With predictive caching, we use an optimization model to automatically decide which objects will be placed in the cache
- Caching the last object as in LRU may be fast but is not always optimal

**LRU example**

Time 1: object 5 requested, cache miss, object 5 is cached, object 1 ejected.

Time 2: object 6 requested, cache miss, object 6 is cached, object 2 ejected.

Time 3: object 2 requested, cache miss, etc.

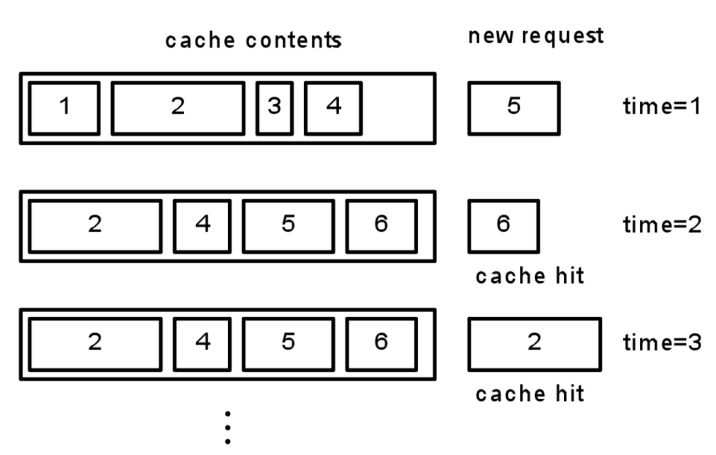



**Predictive caching example**

Time 1: object 5 requested, cache miss. Optimization model decides to cache objects 5 and 6, and eject objects 1 and 3.

Time 2: object 6 requested, cache hit. Optimization model decides not to make any changes in the cache at this time period.

Time 3: object 2 requested, cache hit, etc.




## The mathematical optimization model

- The cache management problem is modelled as a combinatorial optimization problem
- Solution of the problem provides the optimal cache allocation strategy

**Input parameters**

- $s_i$ the size of object $i$
- $C$ the size of the cache
- $c_{1i}$ the cost (latency) to retrieve object $i$ from the shared cache
- $c_{2i}$ the cost (latency) to retrieve object $i$ from the server
- $c_{3i}$ the cost to cache object $i$
- $p_i$ the probability that object $i$ will be requested.

**Decision variables**

$$x_i = \begin{cases} 1, \text{if object } r_i \text{ is placed in the cache,} \\ 0, \text{if object } r_i \text{ is not placed in the cache.} \end{cases}$$

**Optimization model**

$$\min \sum_i c_{1i} p_i x_i + c_{2i} p_i (1 - x_i) + c_{3i} x_i$$

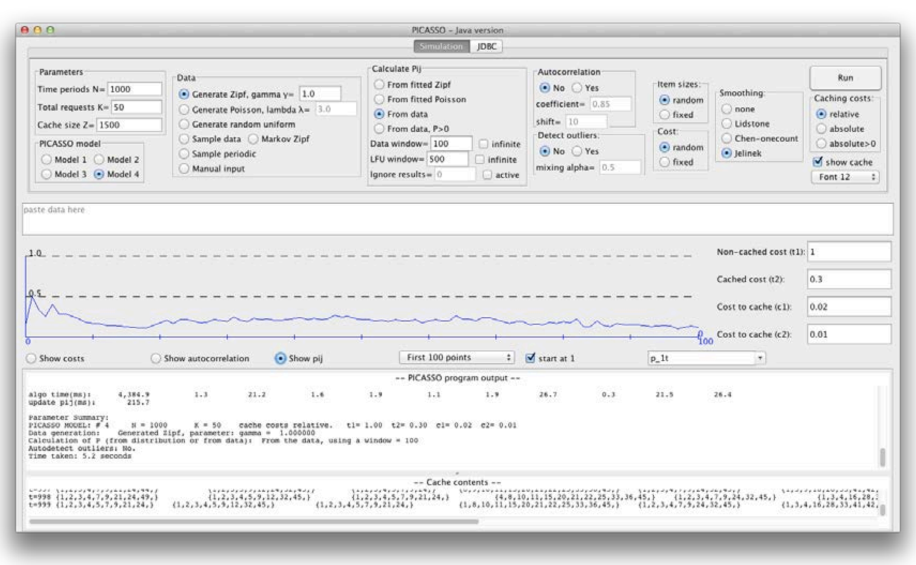
subject to:

$$\sum_i s_i x_i \leq C$$

and $x_i \in \{0,1\}$

## Evaluation results

- Tested the caching framework in simulation scenarios
- The proposed method delivers up to 46% more cache hits and 43% smaller costs compared to the LRU algorithm
- Similarly, the optimized caching algorithm produces up to 26% more cache hits and 30% smaller costs compared to LFU



| Data generation | Optimized predictive caching | Least recently used (LRU) | Least frequently used (LFU) | Belady theoretical optimum |
|---|---|---|---|---|
| Zipf γ=1 | 5748 | 3927 | 4551 | 6112 |
| Zipf γ=1.5 | 8320 | 7418 | 7466 | 8427 |
| Zipf γ=2 | 9516 | 9076 | 9008 | 9458 |
| Zipf γ=3 | 9957 | 9912 | 9805 | 9945 |
| Zipf γ=4 | 9990 | 9977 | 9952 | 9982 |

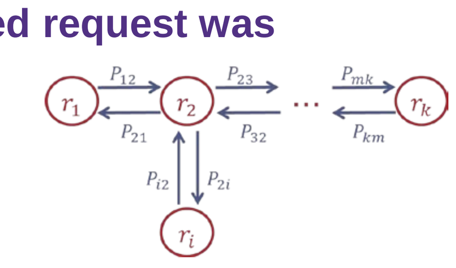
Number of cache hits during 10000 data requests

Average cost per cache hit during 10000 data requests

| Data generation | Optimized predictive caching | Least recently used (LRU) | Least frequently used (LFU) | Belady theoretical optimum |
|---|---|---|---|---|
| Zipf γ=1 | 154.3 | 271.5 | 221.3 | 138.7 |
| Zipf γ=1.5 | 66.9 | 87.9 | 86.8 | 65.1 |
| Zipf γ=2 | 62.9 | 70.3 | 71.6 | 63.6 |
| Zipf γ=3 | 39.8 | 40.4 | 42.0 | 40.0 |
| Zipf γ=4 | 59.2 | 59.4 | 59.9 | 59.4 |

## Calculating the cache object probabilities

- The probabilities in the predictive model need to be recalculated periodically to ensure they are up-to-date
- This can be done using two methods:
  - if the distribution is known or can be guessed, we can fit the distribution to historic data and calculate the object request probabilities from the distribution formula
  - if the distribution is unknown, we can estimate the access probabilities from historic data by calculating the likelihood that a particular object was accessed given what the last observed request was



heig-vd

Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud


Dr Efstratios Rappos
Professor Stephan Robert

IICT - CETT
HEIG-VD
Switzerland


Hes·so

Haute Ecole Spécialisée de Suisse occidentale

Fachhochschule Westschweiz

University of Applied Sciences Western Switzerland