Mistreatment in Distributed Caching Groups: Causes and Implications

Nikolaos Laoutaris[†‡], Georgios Smaragdakis[†], Azer Bestavros[†], [‡]Ioannis Stavrakakis

[†]**Boston University**

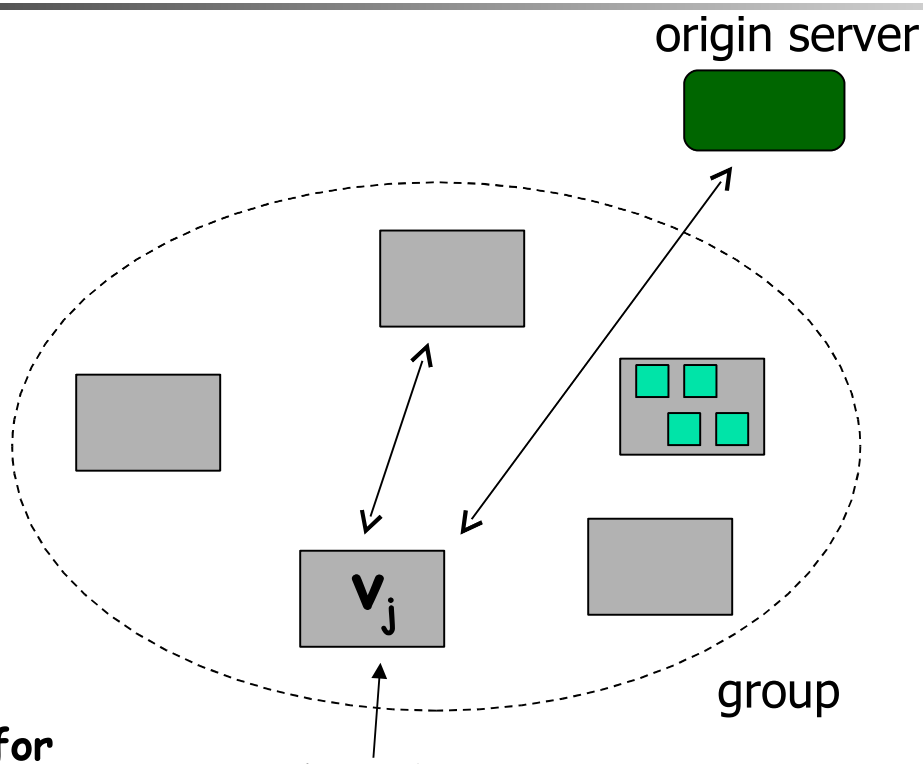
[‡]**University of Athens**

INFOCOM 2006 - Barcelona

This work is about 2 things:

- a new application - **Distributed Selfish Caching (DSC)**
  - distributed caching of web/P2P content under selfish nodes
  - we want to protect DSC against (1) isolationism (2) mistreatment (abuse) and do that in a non-stationary environment (non-fixed operating parameters)
- and a new approach to handling local utility aware nodes – **Self-preservation**
  - *Main idea:* ***"we can borrow concepts and connotations from Game Theory (GT), without using the theory itself"***
  - *Main goal:* ***"to design complex systems that include selfish agents without being constrained in a GT framework"***
  - **An "engineering approach" to handling selfishness**

A distributed replication/caching group



Applications

- Content distribution
- Shared memory
- Network file systems

a collection of
- **n** nodes
- **N** objects

nodes storage used for either replication or caching

replication → **permanent copies**

caching → **temporary copies**

each node
- local client population
- generates requests (preference profile over the N objects)
- requests serviced by the "closest copy"
- fixed amount of storage capacity

Node selfishness brings a new perspective

- the **traditional approach**:
  - entire group under common control
  - find replication/caching strategies to minimize the access cost of the entire group
- but a **selfish node**:
  - wants to minimize (or guarantee some level) the access cost of local users only
  - better model for applications with:
    - multiple/independent authorities
    - e.g., P2P, distributed web-caching
- Therefore, **two new research questions**:
  - "object replication under selfish nodes?" (done, TPDS'06)
  - "object caching under selfish nodes?" (topic of this talk)

Replication strategies – Brief review

- At the two extremes:
  - Socially Optimal (SO) replication (min access cost – entire group)
  - Greedy Local (GL) replication (min access cost – unique isolated node)

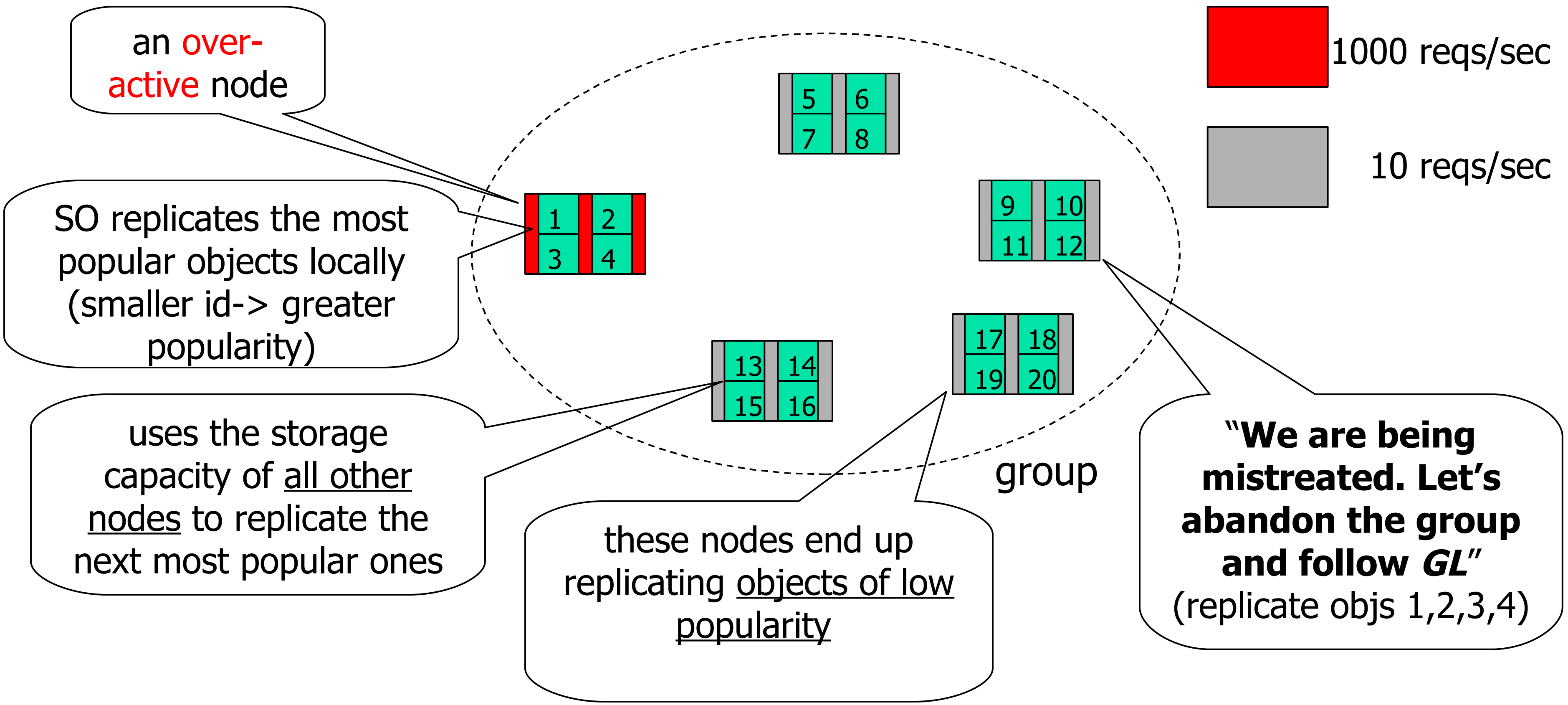
- Both have problems under selfish nodes:
  - SO can lead to mistreatment phenomena

DEFINITION: **A node is being mistreated, if its participations in the group:**

- **leads to a higher (local) access cost**
- **than the minimum one it can guarantee for itself by operating on its own**

  - GL leads to isolationism (uncooperative selection of objects that typically yields poor performance, both local and social)

Mistreatment under SO



nodes go GL and the group disintegrates (everybody replicating 1,2,3,4 is ineffective…)

EQ replication: Our (pure Nash) equilibrium strategy for selfish replication (TPDS'06)

- some nice properties of EQ replication:
  - guarantees a local cost that is lower than GL, <u>for all nodes</u>
  - therefore it precludes mistreatment
  - also, good social cost in typical situations (e.g., under common preference)
  - and finally, low communication requirements for implementation (can use Bloom filters)

BUT…

- all is done for a stationary group
  - <u>we assume everything to be fixed</u>:# nodes, storage capacities, communication costs, object preference profiles

Can we handle dynamic groups?

- *Classic approach*: periodically re-compute EQ
  - how often ???
  - can we gather the necessary input??? (e.g., estimate the local popularity profiles)
- Our new approach: fundamentally different in two ways:
  - Application-wise: uses (on-line) caching instead of (off-line) replication to adaptively track group dynamics
  - Methodologically:
    - We emancipate from GT (we have used pure Nash as **the means** to get what we want. We DO NOT need to make it **the objective**)
    - Instead, we propose an ad hoc "self-preservation" approach to avoid the aforementioned limitations of the classic approach and protect against uncooperativeness and mistreatment in a dynamic group

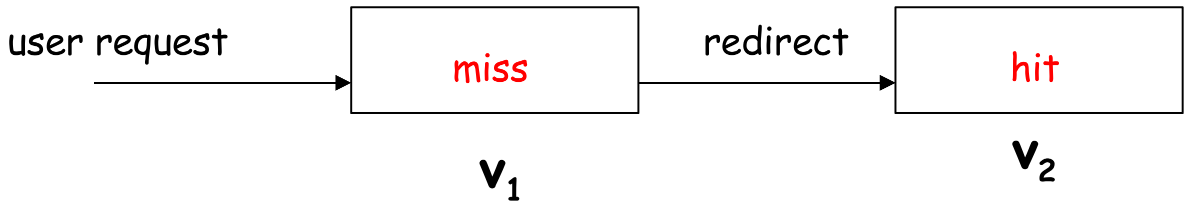
How can mistreatment occur under caching?

- can occur due to *state interaction*
  - cache contents can be affected by the so called "remote hits"

- or due to a *common caching scheme* at all nodes
  - nodes can have different characteristics (be non homogeneous)
    - different capacities, distances to other nodes, etc.

  - and thus a common parameterization of the caching protocol (caching behavior) cannot always perform well for all nodes
    examples coming next

1st case: State interaction



?

- if v_2 discriminates between local and remote hits:
  - local (caching) state NOT AFFECTED
  - requested object sent back to v_1

- else, local state is affected, e.g.:
  - LRU → bring the requested object to the top of the list
  - LFU → increase the frequency count for this object

- v_2's storage can be "hijacked" → access cost increasing for local users

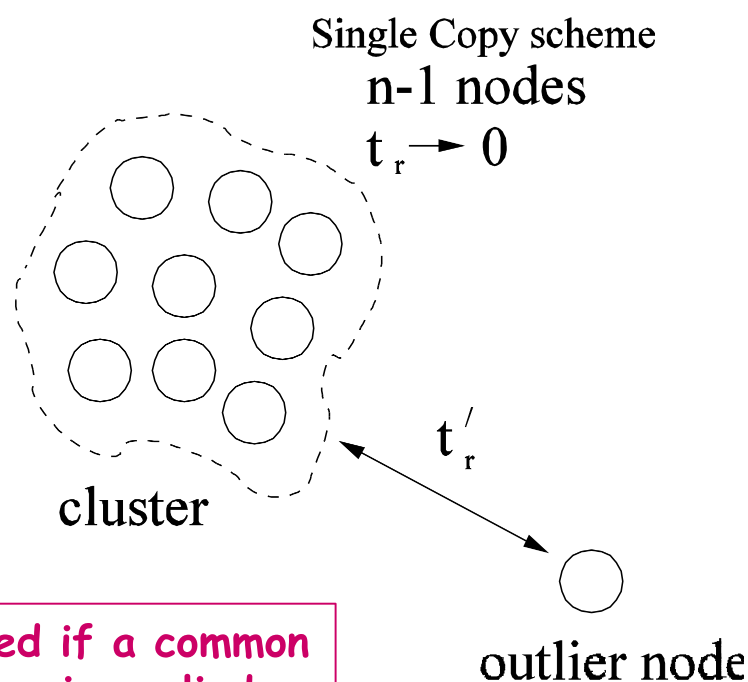
Some facts about state interaction

- High request <u>rate imbalance</u> for mistreatment to appear
  - not easy to do low rate (high potency) attacks (leech without being detected)
- Caching more robust to mistreatment than replication
  - mis. occurs "earlier" under replication (i.e., with smaller imbalance)
  - stochastic nature of caching
- LFU more robust to mistreatment than LRU
  - the higher replacement "noise" of LRU makes it more vulnerable
- Robustness disappears when operating in L2 caching mode
  - proactively fetching objects for remote nodes

2nd case: Common caching scheme

- **Key idea**: a common parametarization across all (dissimilar) nodes can mistreat some
- **Example**: object admission control scheme in an "outlier" node

clustered nodes:
want to avoid multiple copies of the same object
(ideally they would do HASH caching)



outlier: cannot fetch objects efficiently from the cluster
(ideally it keeps copies of all incoming objects)

Either one gets mistreated if a common object admission scheme is applied

A general plan for self-preservation

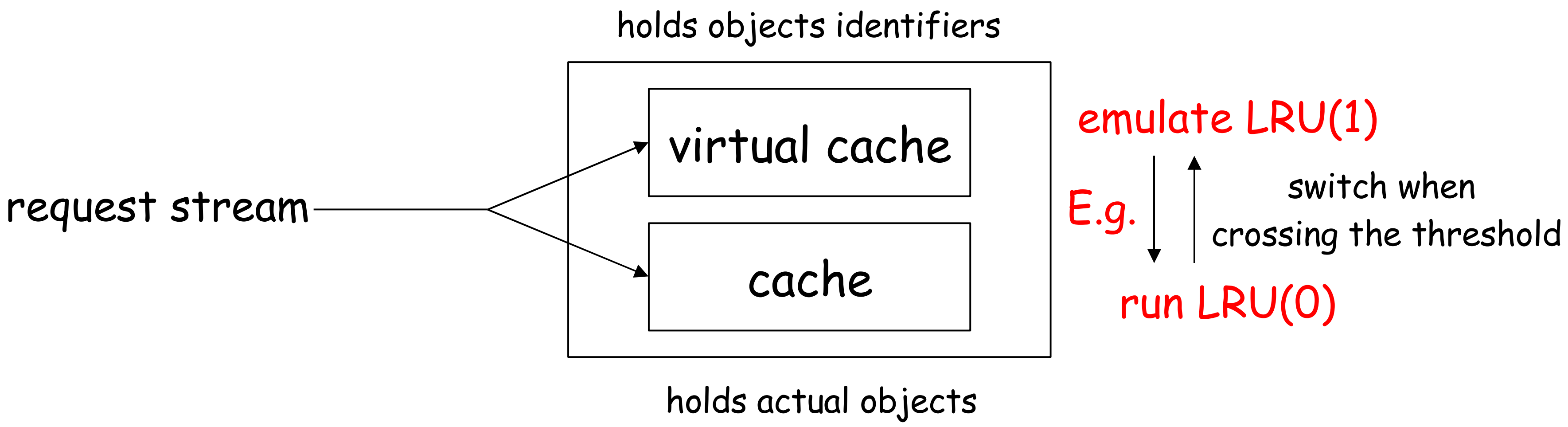
1. Do not avoid cooperation
   1. be open to resource sharing (storage here)
   2. offer to help (send objects back)
2. Keep an open eye for mistreatment by monitoring your running utility (or cost)
3. React when necessary (protect your resources)

- Requirements:
  - be able to detect mistreatment (not trivial in an on-line distributed setting)
  - have an anti-mistreatment device on the side
  - be able to modulate the device depending on the (non-stationary) environment

Making it specific to DSC and the common scheme problem

- Mistreatment detection => Use emulation
  - Virtual cache emulating an alternative caching behavior
- Anti-mistreatment dev. => Object admission control
  - LRU(*q*): keep local copy with probability *q* if object exists in other node (*q*=1 if fetched from the origin server)
- Modulating the device => Change *q* to adjust to current group settings
  - e.g., the "outlier": decreases *q* when getting closer to the cluster, increases it otherwise
  - in this paper: "hard switch" approach between LRU(0) and LRU(1)
  - in a forthcoming one: PID controller for a finer control of *q*
- Can design a similar solution for the state interaction problem

Block diagram



Wrapping it up …

- In a DSC setting we can:
  - protect nodes from mistreatment
  - without resorting to isolationism
  - and do that under a dynamic setting

- Our self-preservation based solution more flexible than our previous GT based approach:
  - is on-line (no need for a priori knowledge or stationarity)
  - is strongly distributed (runs only on local information):
    - updating the local utility as requests get serviced
    - modulation based on local "test for mistreatment" (e.g., emulated virtual cache)

Thank you

Q ?