Metode avansate în sisteme distribuite (Seria: AAC)

Pagina mea ► Cursurile mele ► Master ► Anul 2 ► Semestrul 1 ► M-2-MAS-AAC ► General ► Challenge 4: Distributed Hash Table

Challenge 4: Distributed Hash Table

Implement a best-effort distributed hash table using memcached servers.

- Everything has to be implemented client-side. Your client must support two operations: put and get. All objects must have a replication factor of 2.
- You are expected to use consistent hashing [1] to distribute the objects as evenly as possible across
 the servers.

A few constraints and assumptions:

- · Memcached nodes can run out of memory
- · Memcached nodes can crash
- · Clients are stateless and can't talk to each other
- · Each client knows the identity of all the servers

Test the performance of the system you have built on a 10-node cluster:

- Read/write throughput for objects of different sizes
- Object distribution across nodes (number of objects stored as a function of node rank)
- · Object availability as a function of the number of crashed servers

Implementation hints:

- The DHT is best-effort; in other words, ensuring object availability is attempted, but not guaranteed.
 - It's ok if it is biased toward objects that have recently been written.
 - It's also ok if some tradeoffs are made in favor of performance.
- The clients are stateless; monitoring the servers' availability and keeping track of what gets written where are out of the question. Server crashes (and reboots) can result in data loss.
- Do not use libmemcached's caching disciplines.
- While we suggest a small number of servers for testing purposes, your solution MUST be scalable.
- Use two different hash functions to map keys onto the ring (libcrypto features MD5, SHA etc). It is up to you to decide how to handle collisions (if at all).
- If a node is down, you can assume the following:
 - For the purpose of writing: it will probably be down in the forseeable future (and its successor should be used instead).
 - For the purpose of reading: it was probably down when someone attempted to write the object.
- Think it through: what sensible assumptions can be made if the node is up, but the data is not there?
- In some cases, the client will have to try to talk to more than one server in order to read an object. In what order should the (relevant) servers be contacted?

- To sum it up:
 - When writing an object: What servers do I place it on? What if one (or both) of them are down?
 - When reading an object: Who do I contact (and in what order)? If I can't find the data, when do I stop looking for it?

Testing hints:

- Use Mininet to create a 10-node cluster (plus a node for the client). Don't forget to rate-limit the interfaces!
- It's best if the experiments regarding object distribution are just simultations. You can add a "dry run" mode to your client that just prints out the servers it would place the object on if no failures occur.

Submit a tarball containing your source files, graphs, as well as a README file that explains the structure of the code, how to test it, the main ideas, etc.

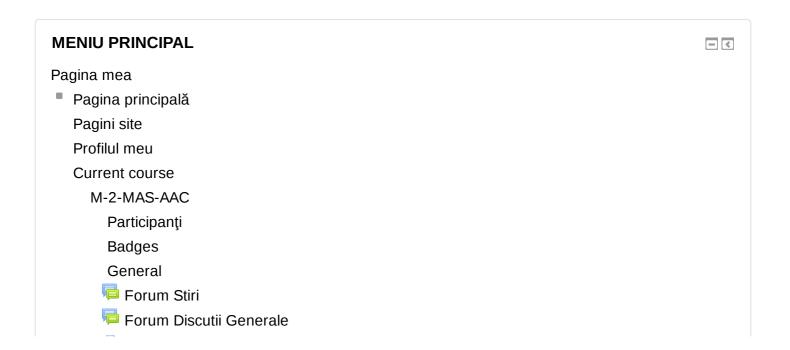
[1] http://en.wikipedia.org/wiki/Consistent_hashing

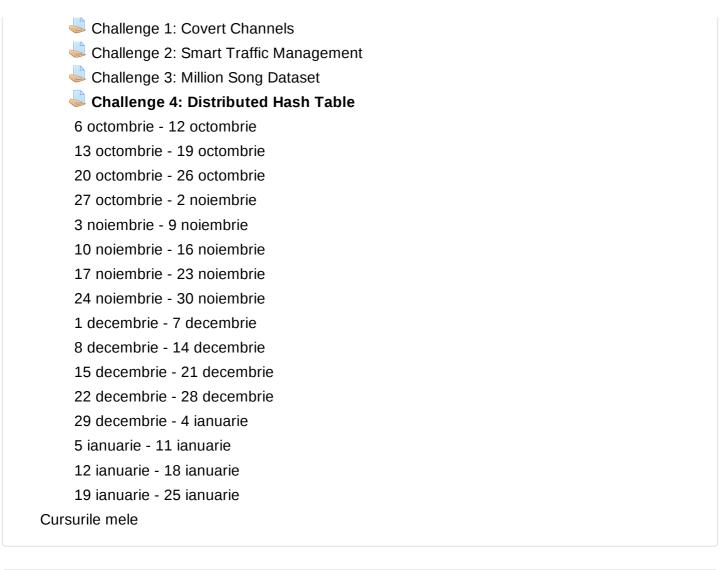
Submission status

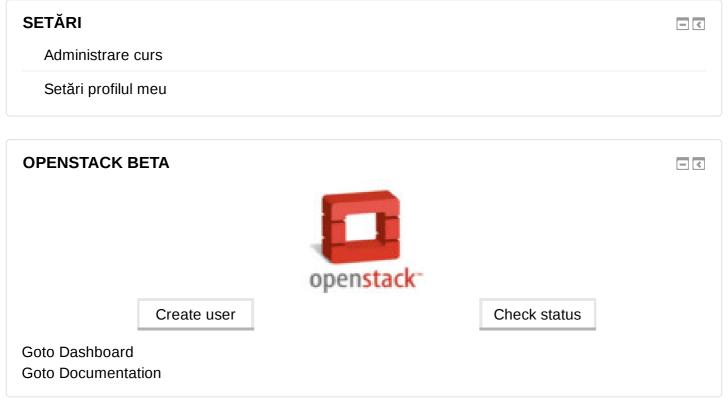
Submission status	No attempt
Grading status	Not graded
Termen predare	luni, 19 ianuarie 2015, 23:55
Time remaining	4 zile

Add submission

Make changes to your submission







Acest site este hostat pe platfoma hardware achitionata din proiectul nr. 154/323 cod SMIS - 4428, "Platforma de e-learning si curricula e-content pentru invatamantul superior tehnic". Pentru mai multe detalii

vezi http://www.curs.pub.ro.

Sunteţi autentificat ca Constantin-Claudiu GHIOC (leşire) M-2-MAS-AAC