FUNCTIONAL SPECIFICATION FOR AN EFFICIENT WAIT-FREE RESIZABLE HASH TABLE PROJECT

Tal Gelbard – 312530371 – talgelbard@campus.technion.ac.il
Alon Libling – 307886192 – lalon@campus.technion.ac.il

OVERVIEW

As the core count of modern processor raises, so does the need of software data structures that can utilize the cores to their full extent. In this project we will implement a wait-free resizable hash table based on this paper. The algorithm in use revolve around using the P-sim global constructor in multiple smaller instances, and the idea of hierarchy-based hash-table, when the resizable object is only the directory itself, thus keeping the look-up time constant.

The paper at hand shows that when the hash-table directory is in stable state, meaning that the resize operation doesn't occur often ($1{:}9 \ \ insert{:}lookup$) , the implementation surpass any other available implementation by quite a lot. This characteristic is useful to many software applications like those who provide searching as a main feature for example: address searching (Google maps), public transportation searching (Moovit) etc.

MAIN FUNCTIONAL ENTITIES

The functional entities are CPP programmers, using the following API:

- *Insert* (also updates the value of an existing key).
- *Look-up*.
- *Delete*.

MUST HAVE FEATURES

- Wait-free: look-up operations will be allowed without any synchronization since the look-up is the most common action.
- Resizable: the data structure size will be dynamic in accordance with the input size.
  - Update operations (insert and delete) on different buckets will be executed in parallel when no resizing actions being handled.
- Unit testing and good debugging framework.

NICE TO HAVE FEATURES

Operations such as:

- Visual printing of the data structure using the buckets.
- Get all keys/values, get size. Return all keys that comply a certain condition.
- Convert to set (using a tree).
- Data cache for faster allocations.

OUT OF SCOPE FEATURES/ISSUES

- Special operations that are been executed on all items:
  - Receives *std::function* or *generic lambda* and activates it on all the items.
- Shrink table size by eliminating the least requested keys operation.
- Nested buckets in order to maintain a smaller DState objects.
- Epoch-based non-blocking garbage collector