Рефакторинг SMR-алгоритма cds::gc::DHP

DHP – это вариант алгоритма Hazard Pointer с неограниченным числом hazard pointer'ов для потока. Текущая реализация (libcds 2.1.0) предполагает, что массив retired data (данных, готовых для удаления) один для всех потоков, то есть все потоки добавляют в него данные и по достижении некоторого предела вызывается основная процедура алгоритма HP – `scan()`, которая проходит по массиву retired ptr и физически удаляет те данные, которые не объявлены как hazard pointer. Очевидно, что `scan()` не должна приостанавливать работу всех остальных потоков; с другой стороны, пока `scan()` работает, массив retired ptr недоступен для изменений, то есть потоки не должны в него добавлять данные.

Эта коллизия разрешается в libcds 2.1.0 следующим образом: имеется циклический список массивов retired ptr, один из которых является текущим. По заполнении текущего массива вызывается `scan()`, которая приватизирует текущий массив retired ptr и объявляет текущим следующий массив из списка. Таким образом, пока `scan()` работает с одним массивом retired ptr, остальные потоки добавляют retired data в другой массив, текущий. Размер циклического списка задается в конструкторе объекта DHP.

Такой алгоритм чреват ABA-проблемой, когда у нас число потоков много больше размера циклического списка.

Варианты решения:

1. Объявить массив retired ptr приватным (thread local data) для каждого потока, работающего с DHP-based структурами данных. При этом надо учитывать:

- размер массива retired data должен изменяться динамически (он зависит от числа hazard ptr на момент вызова `scan()`; критерий увеличения размера: `scan()` не смогла удалить ни одного указателя из retired data).
- нагрузка на структуру данных может быть неравномерна: одни потоки — deleter thread – только удаляют данные из DHP-based структуры данных, то есть активно работают с retired ptr array, другие потоки — updater thread – в основном добавляют/ ищут в структуре. Поэтому retired ptr array каждого потока должен иметь некий `timestamp` последнего применения `scan()` и последующий вызов `scan()` должен происходить не только по заполнении retired ptr array, но и по превышению этого `timestamp` на некоторую дельту.

Это решение плохо тем, что память используется нерационально: потоки, мало удаляющие из структуры данных, будут долго заполнять свой retired array, редко вызывать `scan()`.

2. Retired array - один для всех потоков. Когда один из потоков при вызове `retire_ptr()` обнаруживает, что retired array полон, он приватизирует retired array и вызывает `scan()`. При этом поток должен создать новый retired array, чтобы остальные потоки могли продолжить свою работу. Размер retired array в принципе не ограничен. Если `scan()` обнаруживает, что может удалить (delete), например, не более половины элементов из текущего retired array, она должна увеличить размер массива, то есть увеличить порог срабатывания следующего вызова `scan()`. Получается, что, ради эффективности, “массив” retired array должен быть не массивом, а списком блоков. Блоки берутся из некоторого пула блоков, конечно, в lock-free манере. Следовательно, нужно решить ABA-проблему для этого пула.

Попутно следует упростить реализацию DHP, сократив иерархию структур.

Требование: API (public interface and nested classes) DHP-алгоритма измениться не должно.