Project Proposal

Description:

Implement the project idea sorting algorithms benchmark and implementation (2018)

Applicant Information:

Name: Kefan Yang

Country of Residence: Canada

University: Simon Fraser University

Year of Study: Third year

Major: Computing Science

Relevant experience:

Mini SQL – Database System Course Project

A simple SQL database supporting basic SQL syntax like ‘select’, ‘insert’, ’delete’ and ‘update’ Implemented a disk-based b+ tree to index data records

GitHub: https://github.com/ADS-DBAwesomeGroup/MiniSQL

The 2017 ACM-ICPC Pacific Northwest Regional Contest

Familiarity with basic sorting algorithms and deep understanding of traditional quick sort

Deliverables:

1. A benchmark for sorting algorithms

2. Benchmark implementation for the sorting algorithms mentioned in research papers.
3. The winner under the given benchmark
4. Industrial implementation for the selected sorting algorithm in **pg_qsort()**, **pg_qsort_args()** and **gen_qsort_tuple.pl**
5. New hash implementation
6. Review of other's patches

TimeLine:

Community Bonding Period (April 23 – May 14)

- Get in touch with mentors and other community members. Try to get suggestions on this proposal as soon as possible.
- Get more familiar with codebase. Fully understand the usage of **pg_qsort()**, **pg_qsort_args()** and **gen_qsort_tuple.pl**.
- Discuss on how to implement the benchmark – what is the "average" use case?
- Read research paper

Week 1 (May 14 – May 20)

- Benchmark implementation of IntroSort

Week 2 (May 21 – May 27)

- Benchmark implementation of TimSort

Week 3 – 4 (May 28 – June 10)

- Benchmark implementation of QuickSort and RadixSort (this should not take much time)
- Implement benchmark (Phase 1 and 2)

First Evaluation Period Deliverables

1. Benchmark implementation of IntroSort, TimeSort and RadixSort
2. Phase 1 and 2 of the benchmark implementation

Week 5 (June 11 – June 17)

- Review feedbacks from the first evaluation and make some changes accordingly
- Adjust plans for the following weeks if necessary

Week 6 (June 18 – June 24)

- Test the sorting algorithms under phase 1 and 2
- Phase 3 of the benchmark implementation

Week 7 – 8 (June 25 – July 8)

- Select the winner and implement its industrial version for **pg_qsort()** and **pg_qsort_args()**

Second Evaluation Period Deliverables

1. Complete benchmark implementation
2. Winner algorithm and some analysis
3. New implementation of **pg_qsort()** and **pg_qsort_args()**

Week 9 (July 9 – July 15)

- Review feedbacks from the second evaluation and make some changes accordingly
- Industrial implementation for **gen_qsort_tuple.pl** (may encounter some challenges with Perl script)

Week 10 (July 16 – July 22)

- Test new hash table implementation
- Industrial implementation of hash table

Week 11 – 12 (July 23 – Aug 6)

- Final buffer period
- Review other's patches
- Prepare for the final submission

Implementation

Benchmark:

After the discussion with the mentors and other community members, I had a new design of benchmark implementation. Basically, I will evaluate the performance of a sorting algorithm through three phases:

Phase 1: Test on random arrays

The main purpose is to evaluate the average performance of the sorting algorithms. Test cases will be generated using different array sizes and data types (integers, strings and tuples). The performance of each algorithm is measure in CPU clock cycles.

Phase 2: Test on worst case

In this case, I plan to use pre-generated integer arrays to test the worse case performance of each algorithm. The worst-case scenario for each sorting algorithm is listed below:

QuickSort: Median of three killer sequence

IntroSort: IntroSort use quicksort at the beginning, so its worst-case scenario should be very similar with QuickSort.

TimSort: Still working on it

RadixSort: According to how this algorithm works, it should have a very bad performance with very large dataset. (RadixSort consumes a lot of memory, which may cause page fault when QuickSort only get cache misses)

Phase 3: Test on SQL statement

After we get a winner from the first two phases, I am going to test it using SQL statements and compare it with the QuickSort implementation. I think the current pgbench implementation will be enough for this. But I will change the proportion of each type SQL statements to simulate different scenarios.

Benchmark implementation of sorting algorithms:

Implement the pseudo-code provided by the research papers in C.

QuickSort: Simply use the Bentley & McIlroy implementation.

IntroSort: There is a highly optimized cpp implementation in STL (generally 20% faster than hand-coded cpp implementation). I will try to implement it in C to see if there's a performance gain.

TimSort: There's a Java implementation, which I am going to rewrite in C.

RadixSort: C implementation is given in the paper.

Industrial implementation of selected sorting algorithm:

The industrial version is basically an optimization based on the benchmark implementation. Some of the optimizations I come up with now are:

1. Check if input array is already sorted
2. Applying insertion sort directly for short arrays.
3. Convert recursions to iterations.
4. Optimize loops to avoid potential cache misses.

But I think the method here highly depends on the actual implementation of that algorithm and it's difficult for me to decide in the proposal stage.

New hash implementation:

For cuckoo hashing, I plan to use some code from other open source implementation to test its performance. But I guess finally I need to implement it myself because the code in most of these projects is far from optimized.