



ΕΤΟΣ: 2020-2021


1Η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ -ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΜΑΡΙΑ ΗΛΕΚΤΡΑ ΓΑΡΓΑΛΑ – Α.Μ.4330

ΔΗΜΟΣΘΕΝΗΣ ΖΑΓΚΑΣ – Α.Μ.4359

Email: cs04330@uoi.gr

Email: cs04359@uoi.gr



Γενική περιγραφή προγράμματος:

Το project που μας ανατέθηκε, υλοποιεί μία μηχανή αποθήκευσης δεδομένων (key-value store). Βασίζεται στην δομή LSM-Tree, μία δομή που διατηρεί ταξινομημένα τα δεδομένα εσωτερικά στην μνήμη και στον δίσκο, και αποθηκεύει και αναζητά ζευγάρια κλειδιού-τιμής. Τα ταξινομημένα δεδομένα βοηθούν στην πολυπλοκότητα, γιατί μπορούμε να τα εγγράψουμε και να τα αναζητήσουμε πιο γρήγορα.

Το LSM-Tree αποτελείται από δύο δομές, την Memtable και την SST. Η δομή Memtable υλοποιείται με μία δομή δεδομένων με πολλές λίστες και βρίσκεται πάντα στην μνήμη. Ουσιαστικά, μία κλήση για εισαγωγή κλειδιού-τιμής (`db_add()`) εισάγεται στο Memtable. Αν κάνω κλήση της `db_get()` αναζητά αρχικά την εγγραφή στο Memtable και εφόσον δε βρεθεί θα την αναζητήσει στο SST. Σε περίπτωση που το Memtable γεμίσει, θα γίνει merge (στον δίσκο) για να μεταφερθούν τα αρχεία στο δίσκο.

Στην Memtable υπάρχει κι ένα αρχείο log όπου αποθηκεύει στον δίσκο τις εγγραφές. Η ύπαρξη του αρχείου log, χρησιμεύει σε περιπτώσεις σφαλμάτων, παραδείγματος χάριν όταν κρασάρει ο υπολογιστής μας ή κλείσει καταλάθος.

Η εντολή `put` εισάγει στο Memtable και στο log αρχείο ένα ζευγάρι κλειδί-τιμή στην βάση. Σε περίπτωση που το log αρχείο γεμίσει, το Memtable παγώνει, το γράφουμε στο δίσκο και φτιάχνουμε ένα καινούριο Memtable για τις επόμενες εγγραφές.

Η δομή SST βρίσκεται στον δίσκο και έχει πολλά επίπεδα (έξι), όπου το καθένα από αυτά περιέχει αρχεία με ταξινομημένες εγγραφές. Σε περίπτωση που τα αρχεία οποιουδήποτε επιπέδου γίνουν πάρα πολύ μεγάλα ή όταν το πλήθος των αρχείων είναι πολύ μεγάλο, γίνεται συγχώνευση (combaction) και το αρχείο αυτό πηγαίνει στο επόμενο επίπεδο.

Η εντολή `get` αναζητά πρώτα στο Memtable τις εγγραφές και αν δεν βρεθούν, τότε τις αναζητά στο SST (σε όλα τα επίπεδα: level0-level6).

Υλοποίηση

Γενική περιγραφή:

Στόχος είναι η παραλληλοποίηση των functions `_write_test` και `_read_test` το οποίο επιτυγχάνεται με την δημιουργία νημάτων στην `bench.c`, καθώς και ο παραλληλισμός της `db_get()`.

Αναλυτική Εξήγηση:

Αρχικά ζητούμε στην γραμμή εντολών τον αριθμό των αιτήσεων για `read` και για `write` και τον αριθμό των νημάτων που επιθυμεί ο χρήστης. Ο χρήστης μπορεί να επιλέξει και τις λειτουργίες `write` ή `read`, γράφοντας και τον αριθμό των αιτημάτων για την λειτουργία που επέλεξε (`./a.out write 1000` ή `./a.out read 300`).

Παράδειγμα: `./a.out read_write 600 50 35` : επιλογή `read_write`, 600 αιτήσεις για `read`, 50 αιτήσεις για `write` και 35 νήματα. Με βάση τον αριθμό των αιτήσεων για τις λειτουργίες `read` και `write` χωρίζουμε και τα νήματα, ώστε να βελτιώσουμε την απόδοση. Όταν δηλαδή έχω 9000 `write` και 1000 `read`, θα χωρίζουμε τα νήματα ώστε να εκμεταλλευτεί τα περισσότερα η `write`, που έχει τον μεγαλύτερο αριθμό αιτήσεων. Αυτό έχει σκοπό την βελτιστοποίηση του χρόνου, αφού είναι λογικό η συνάρτηση με τις περισσότερες αιτήσεις να έχουν ανάγκη περισσότερα νήματα.

Bench.h:

- Γραμμή 8-12: Προσθήκη `#include <pthread.h>` & `#include "../engine/db.h"` & `#define DATAS ("testdb")`. Ο λόγος εξηγείται παρακάτω.
- Γραμμή 25-27: Αρχικοποίηση των πρωτότυπων συναρτήσεων για `_write_test`, `_read_test` και `statistics` (θα εξηγηθεί η χρήση της παρακάτω)
`void statistics(void * temp,int flag);`
`void *_write_test(void * args_write);`
`void *_read_test(void * args_read);`
- Γραμμή 30-36: Ορισμός του struct. Οι μεταβλητές της δομής είναι `long int count = long int write_count; + long int read_count;` και `int r;` και `DB* db;` Οι παράμετροι των `_write_test` και `_read_test` τροποποιήθηκαν κατάλληλα λόγω της χρήσης της `pthread_create(..)`. Πλέον οι παράμετροί τους εισέρχονται μέσω της struct που δημιουργήσαμε.

```

struct args{
    long int count;
    long int write_count;
    long int read_count;
    int r;
    DB* db;
};

```

Bench.c:

- Γραμμή 21: Προσθήκη της printf(numOfThreads) στο print_header για να τυπώνεται ο αριθμός των δοθέντων νημάτων στην περίπτωση που ο χρήστης έχει επιλέξει _read_write. Επίσης, η _print_header παίρνει σαν παράμετρο από την κλήση της και το numOfThreads.

- Γραμμή 73-83: Ορισμός των απαραίτητων μεταβλητών.

```

DB* db; int flag;

long int numOfThreads_write;

long int numOfThreads_read;

long int numOfThreads;

long int read_count;

long int write_count;

long int ceiling;

long double num;

struct args thread_args;

thread_args.r = 0;

```

- Γραμμή 73: Ορισμός της db. Η βάση θέλουμε να ανοίγει/κλείνει μία φορά κι όχι από κάθε thread, γι' αυτό και την απομονώσαμε από την kiwi.c και καλείται πλέον στην bench.c. Η db ανοίγει μία φορά (γραμμές 154,174,188) και στα τρία if ξεχωριστά (για read, write και read_write) και τυπώνει τις λεπτομέρειές της, ενώ κλείνει μία φορά στο τέλος του προγράμματος (χρειάστηκε να κάνουμε #include "../engine/db.h" και #define DATAS ("testdb") στην bench.h).
- Γραμμή 85-89: Έλεγχος με if για το εάν έχει δεχθεί τα κατάλληλα ορίσματα από το terminal.

- Γραμμή 92-94: Έλεγχος των ορισμάτων της main με if για το αν θα εκτελέσουμε read, write ή read_write.
- Γραμμή 95-107: Το argc > 3, άρα ζητάμε να κάνει read_write, οπότε και αρχικοποιούμε όλες τις μεταβλητές από τα ορίσματα της main. Το argv[2] αναφέρεται στις αιτήσεις της read, το argv[3] αναφέρεται στις αιτήσεις της write και το argv[4] στον αριθμό των νημάτων. Ορίζουμε και τις παραμέτρους του struct αντίστοιχα.

```
write_count = atoi(argv[3]);
thread_args.write_count = write_count;
read_count = atoi(argv[2]);
thread_args.read_count = read_count;
thread_args.count = write_count + read_count;
numOfThreads = atoi(argv[4]);
```

- Γραμμή 106-141: Χωρισμός των threads. Θέλουμε να χωρίζονται τα νήματα ανάλογα με τον αριθμό των αιτήσεων. Για παράδειγμα για 60 write και 40 read με 10 νήματα, 6 νήματα θα χρησιμοποιηθούν στην write και 4 στην read.

Μία κρίσιμη περίπτωση όπου δεν γινόταν σωστά ο χωρισμός των νημάτων:

Έχουμε read = 4, write = 8, threads = 4. Ο επιθυμητός χωρισμός των νημάτων θα είναι 3 για write και 1 για read. Παρ' όλα αυτά το ποσοστό $(write/(read+write)) * threads$ είναι 2.9, και λόγω του integer(2.9) είναι 2 και επομένως χάνουμε ένα thread. Για τον αποφυγή του προβλήματος αυτού, αρχικοποιούμε μια μεταβλητή **double num**; όπου δίνεται το ποσοστό των νημάτων για write και παίρνουμε τεχνητά το **int ceiling**; της **num** (χωρίς χρήση της βιβλιοθήκης math.h), το οποίο θα ισούται με 3. Κάνοντας μία αφαίρεση των νημάτων για write από τα ολικά νήματα, παίρνουμε τον αριθμό των νημάτων για read.

Μία άλλη κρίσιμη περίπτωση είναι:

Ο υπολογισμός του ποσοστού να έχει αποτέλεσμα μικρότερο του 1 (πχ 0.3) όπου για integer(0.3) θα γίνει 0, το οποίο δεν είναι αποδεκτό. Σ' αυτή την περίπτωση ορίζουμε 1 thread (πχ για το read) και αφαιρούμε 1 από τα threads για write.

```
if((ceiling - num)<0.5){
    numOfThreads_write = ceiling;
}
else{
    numOfThreads_write = (int)num;
    if(numOfThreads_write == 0){
        numOfThreads_write = 1;
    }
}
```

```

numOfThreads_read = numOfThreads-numOfThreads_write;
if(numOfThreads_read == 0){
    numOfThreads_read = 1;
    numOfThreads_write--;
}
}

```

- Γραμμή 148-161: Βρισκόμαστε στην περίπτωση όπου πραγματοποιείται μόνο η write, `thread_args.write_count = thread_args.count` αφού όλα τα count είναι για το write. Ορίζουμε `flag=0` που θα χρησιμοποιηθεί αργότερα στην statistics όπου και θα αναλυθεί.
Επίσης στην συνάρτηση `_printf_header` περνάμε σαν δεύτερο όρισμα το 0, αφού δεν έχουμε νήματα.

Την μεταβλητή `r`, αφού πλέον την περνάμε από τον struct, την χρησιμοποιούμε στην Γραμμή 156-157 όπως φαίνεται:

```

if (argc == 4){
    thread_args.r = 1;
}

```

- Γραμμή 164-176: Βρισκόμαστε στην περίπτωση για read και πραγματοποιούνται οι αντίστοιχες ενέργειες όπως στην write.
Γραμμή 172: Ορίζουμε την `thread_args.write_count = 0` ώστε να προσπεράσουμε το `pthread_cond_wait` όταν εκτελούμε μόνο read.

- Γραμμή 179-236: Βρισκόμαστε στην περίπτωση `read_write`. Στην Γραμμή 184-185 καλούμε τις συναρτήσεις `_print_header` & `_print_environment` ώστε να εκτυπώνονται οι πληροφορίες για την κεφαλίδα και το σύστημα. Στην Γραμμή 188-190 έχουμε βάλει αντίστοιχη περίπτωση για `thread_args.r = 6`, όπως κάνουμε και στις άλλες δύο περιπτώσεις. Στην Γραμμή 191 κάνουμε δυναμική δέσμευση μνήμης για τα νήματα.

```

newthread = (pthread_t *)malloc(numOfThreads * sizeof(pthread_t));
if(newthread == NULL){ //check if malloc worked
    printf("Malloc failed!\n");
    exit(1);
}

```

Στην Γραμμή 204-231 καλούμε τις συναρτήσεις `_write_test` και `_read_test` μέσω των νημάτων.

Η δημιουργία των νημάτων θέλουμε να γίνει παράλληλα ως έναν βαθμό κι όχι σειριακά (όχι δηλαδή πρώτα όλα τα νήματα για write και μετά όλα τα νήματα για read, σε δύο for). Αντίθετα, θα θέλαμε σε μια for, σε κάθε επανάληψη της, να δημιουργούμε νήματα για write και για read. Αυτό πραγματοποιείται με τον παρακάτω κώδικα, που θα τον εξηγήσουμε με ένα παράδειγμα.

Έστω ότι έχουμε 4 νήματα για write και 6 για read. Δημιουργούμε με μία for παράλληλα τα 4 νήματα για write και read και στην συνέχεια με μία δεύτερη for δημιουργούμε τα δύο υπολειπόμενα νήματα για read.

```
if(numOfThreads_read>numOfThreads_write){
    for(i=0; i<numOfThreads_write;i++){
        pthread_create(&newthread[i],NULL,_write_test,(void *)&thread_args);

        pthread_create(&newthread[i+numOfThreads_write],NULL,_read_test,(void *)
&thread_args);
    }

    for(i=numOfThreads_write; i<(numOfThreads - numOfThreads_write);i++){
        pthread_create(&newthread[i+numOfThreads_write],NULL,_read_test,(vo
id *) &thread_args);
    }
}
```

Το αντίστοιχο συμβαίνει όταν έχουμε περισσότερα νήματα για write αντί για read, ενώ αν είναι ίσα χρησιμοποιούμε μόνο μία for.

Στην Γραμμή 234-237 κάνουμε χρήση της join για να περιμένουμε όλα τα νήματα πριν τερματίσουμε.

- Γραμμή 245: Κλείνουμε την βάση.
- Γραμμή 246: Καλούμε την statistics με ορίσματα το struct thread_args (περιέχει τον αριθμό των write/read, το count και την μεταβλητή db) και το flag.

Kiwi.c:

- Γραμμή 8-9: Ορίζουμε τα mutex_lock.
- Γραμμή 11: Ορίζουμε την μεταβλητή συνθήκης (condition Variable)
- Γραμμή 13-29: Ορίζουμε/αρχικοποιούμε κάποιες μεταβλητές που θα χρειαστούν στην συνέχεια
- Γραμμή 31-106 write_test:
Στην Γραμμή 31, για την δημιουργία νημάτων, φέρνουμε τη συνάρτηση _write_test στην μορφή void* _write_test(void * args). Στην Γραμμή 41-44 δημιουργούμε ένα τοπικό struct args, μέσω του οποίου αντιγράφουμε τις μεταβλητές (count,r,db). Αυτό το κάναμε για να μην αλλάξουμε σε όλον τον κώδικα τις μεταβλητές αυτές με αυτές που ορίζουμε στην bench.

Στην Γραμμή 59-66, για να υπολογίσουμε το συνολικό κόστος χρειαζόμαστε το start από το πρώτο νήμα, που σηματοδοτεί την έναρξη. Χωρίς κάποια τροποποίηση του κώδικα, κάθε νήμα που εκτελούσε την συνάρτηση άλλαζε την τιμή του start. Γι' αυτό με την χρήση αμοιβαίου αποκλεισμού παίρνουμε το πρώτο start μέσω της if και τα υπόλοιπα νήματα δεν μπορούν να το αλλάξουν.

```
pthread_mutex_lock(&mymutexW);  
if(k == 0){  
    startW = get_ustime_sec();  
    k = 1;  
}  
pthread_mutex_unlock(&mymutexW);
```

Την μεταβλητή start την ορίσαμε ως global για να είναι καθολική η τιμή της.

Στην Γραμμή 70 δημιουργούμε το mutex_lock όπου περιμένουν όλα τα νήματα πριν την εκτέλεση του db_add.

Στην Γραμμή 71, κάνουμε αύξηση του global j(counter) για να παίρνει το επόμενο νήμα την ενημερωμένη τιμή.

Στην Γραμμή 72 ελέγχουμε αν το j είναι μεγαλύτερο από το count κι αν ισχύει κάνουμε break (αφού θα έχουμε εκτελέσει όλες τις add), αλλιώς συνεχίζουμε.

Στην Γραμμή 90 γίνεται το db_add.

Στην Γραμμή 91 ελέγχουμε αν το j ισούται με το count· τότε στέλνουμε σήμα ώστε να αρχίσουν την λειτουργία τους όλα τα νήματα (**pthread_cond_broadcast(&condVar);**) στην read (θα εξηγηθεί στην read), και αυξάνουμε το done κατά ένα.

```
db_add(db, &sk, &sv);  
if(j == count){  
    pthread_cond_broadcast(&condVar);  
    done = 1;  
}
```

Στην Γραμμή 105, παίρνουμε από το τελευταίο νήμα τον χρόνο λήξης της write, αποθηκεύοντάς το στην καθολική μεταβλητή **long long int endW;**

Στην Γραμμή 106 παίρνουμε το συνολικό κόστος: **costW = endW - startW;**

- Γραμμή 109-185 `read_test`:
Στην Γραμμή 111, για την δημιουργία νημάτων, φέρνουμε τη συνάρτηση `_read_test` στην μορφή `void* _read_test(void * args)`.
Στην Γραμμή 124-127 κάνουμε την ίδια διαδικασία για την struct με αυτήν της `_write_test`.
Στην Γραμμή 129-137, δέχεται το σήμα από την `_write_test` και η τιμή της `done` παύει να είναι διάφορη του 1, κι έτσι απελευθερώνονται όλα τα νήματα.

```
pthread_mutex_lock(&mymutexW);  
while(done != 1){  
    pthread_cond_wait(&condVar,&mymutexW);  
}  
pthread_mutex_unlock(&mymutexW);
```

Στην Γραμμή 139-145, για την απολαβή του χρόνου εκτελούμε την ίδια διαδικασία που κάναμε στην `write`.

Στην Γραμμή 147, η `while` δημιουργήθηκε παρόμοια με αυτήν της `_write_test`. Η μόνη διαφορά είναι ότι η `unlock(&mymutexR)` γίνεται στην

Γραμμή 168 (πριν από την κλήση της `db_get`) αφού έχουμε βάλει εσωτερικά lock στην `db_get`, όπως θα περιγραφεί στην `db.c`.

Στην Γραμμή 173, γίνεται το `unlock_mutex` του `sst_get` στην `db.c` καθώς το lock του ίδιου mutex γίνεται μέσα στην `db.c`. Το `unlock` δε μπορεί να γίνει μέσα στο `db.c` καθώς το `sst_get` εκτελείται στο `return` του `db_get`.

- Γραμμή 195-230: Υλοποιείται η `statistics` η οποία είναι υπεύθυνη για την τύπωση των στατιστικών για κάθε μία από τις 3 περιπτώσεις (`read`, `write`, `read_write`). Για `flag=0` είναι η περίπτωση `write` και τυπώνονται τα στατιστικά μόνο για την `write`, για `flag=1` είναι η `read` και τυπώνονται τα στατιστικά μόνο για την `read` και για `flag=2` είναι η `read_write` και τυπώνονται τα στατιστικά για την `write` και την `read`.

Γραμμή 198, υπολογίζεται το ολικό κόστος προσθέτοντας το κόστος της καθολικής μεταβλητής `read` και `write`.

```
cost = costR+costW;
```

Στην Γραμμή 196 φτιάχνουμε έναν struct για να πάρουμε τον συνολικό αριθμό των αιτήσεων (count) από τον struct που μας δίνει σαν παράμετρο η bench.c.

```
struct args* stats = (struct args*) temp;
```

- Με βάση τον ορισμό που βρήκαμε στο διαδίκτυο:

Ορισμός ρυθμοαπόδοσης: $R = \frac{1}{t}$, όπου I όγκος των δεδομένων σε bits, t ο χρόνος που απαιτήθηκε για την ολοκλήρωση της μετάδοσης.

Σε κάθε read ή write, μεταφέρουμε προς/από την βάση ένα ζευγάρι κλειδί-τιμή, τα οποία είναι τύπου char, δηλαδή 1byte(=8bit) το καθένα. Επομένως το ζευγάρι είναι 16bits συνολικά. Όταν εκτελούμε read_write, επειδή εκτελούμε και τις δύο συναρτήσεις (read και write), μεταφέρουμε 32bit σε κάθε εκτέλεση.

Άρα, η ρυθμοαπόδοση = $(32 * \text{count}) / \text{cost}$, όπου count είναι ο αριθμός των αιτήσεων (read και write) και cost το ολικό κόστος.

```
long int totThroughput;
```

```
totThroughput = (32*stats->count)/cost;
```

Από την Γραμμή 214-229 κάνουμε τις κατάλληλες αλλαγές στις printf ώστε να τυπώνονται σωστά τα στατιστικά. Η ρυθμοαπόδοση θέλουμε να τυπώνεται μόνο στην περίπτωση που έχουμε read_write (δηλαδή στην πρώτη περίπτωση που έχουμε μόνο flag==2).

- Γραμμή 206-211:

Όταν το κόστος είναι 0 απαιτώντας λίγες αιτήσεις, τότε ορίζουμε την ρυθμοαπόδοση ίση με το μηδέν, για να αποφύγουμε την διαίρεση με το μηδέν (θα τείνει άπειρο).

```
if(cost == 0){
    totThroughput = 0;
}
else{
    totThroughput = (32*stats->count)/cost;
}
```

db.h:

- Γραμμή 10: Ορισμός του pthread_mutex όπου θα χρησιμοποιηθεί στην kiwi.c & db.c.

```
pthread_mutex_t mymutex_get_sst;
```

db.c:

- Γραμμή 8: Αρχικοποίηση mutex

```
pthread_mutex_t mymutex_get_memtable = PTHREAD_MUTEX_INITIALIZER;
```

- Γραμμή 69-84: Η αρχική ιδέα ήταν να κλείναμε ολικά την db_get και να την εκτελούσε ένα νήμα την φορά για αναζήτηση. Αφού όμως οι δύο δομές είναι ανεξάρτητες, κλείνουμε το mutex πάνω από την db_add (όπως φαίνεται στην kiwi.c), ώστε να περνάνε όλα τα νήματα στην συνάρτηση και βάζουμε mutex στην memtable_get και την sst_get για να δημιουργήσουμε καλύτερο παραλληλισμό.

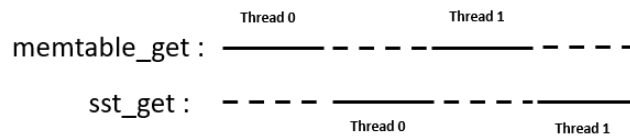
```
pthread_mutex_lock(&mymutex_get_memtable);  
if (memtable_get(self->memtable->list, key, value) == 1){  
    pthread_mutex_unlock(&mymutex_get_memtable);  
    return 1;  
}  
pthread_mutex_unlock(&mymutex_get_memtable);  
pthread_mutex_lock(&mymutex_get_sst);  
return sst_get(self->sst, key, value);
```

Παράδειγμα: Έστω δύο threads t1 & t2 με t1 να περνά από το πρώτο lock. Καλεί την memtable_get ψάχνοντας στην memtable για το κλειδί. Αν δεν το βρει κάνει unlock το mutex της memtable και κλειδώνει του sst. Παράλληλα, αφού το mutex της memtable είναι ξεκλειδωτό, το t2 νήμα επιτρέπεται να εκτελέσει την memtable_get ενώ το t1 νήμα βρίσκεται στην sst_get. Άρα εκτελείται παράλληλη αναζήτηση στις δύο δομές.

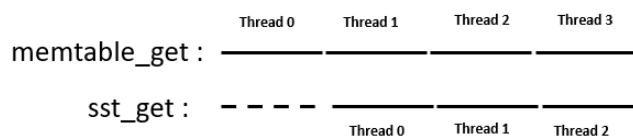
Παρακάτω φαίνεται μια εικονική περιγραφή της αρχικής σειριακής εκτέλεσης των memtable_get και sst_get σε αντίθεση με την δική μας υλοποίηση παραλληλισμού.

Running : _____
Idle : - - - -

Original



Concurrency



- Για την db_add: Προσπαθήσαμε να εισάγουμε παραλληλισμό στην db_add. Παρ' όλα αυτά δεν βρήκαμε δομές, που να μπορούν να εκτελεστούν παράλληλα, αφού οι εγγραφές γίνονται μόνο στην δομή memtable (μπορεί να εκτελεστεί ένα write την φορά).

Εκτέλεση της εντολής *make all* στο φάκελο *kiwi-source*.

```
myy601@myy601lab1:~/kiwi/kiwi-source$ make all
cd engine && make all
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/engine'
CC db.o
CC memtable.o
CC indexer.o
CC sst.o
CC sst_builder.o
CC sst_loader.o
CC sst_block_builder.o
CC hash.o
CC bloom_builder.o
CC merger.o
CC compaction.o
CC skiplist.o
CC buffer.o
CC arena.o
CC utils.o
CC crc32.o
CC file.o
CC heap.o
CC vector.o
CC log.o
CC lru.o
AR libindexer.a
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/engine'
cd bench && make all
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/bench'
gcc -g -gdb -Wall -Wno-implicit-function-declaration -Wno-unused-but-set-variable bench.c kiwi.c -L ../engine -lindexer -lpthread -lsnappy -o kiwi-bench
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/bench'
myy601@myy601lab1:~/kiwi/kiwi-source$
```

Παρακάτω παραθέτουμε παραδείγματα εκτέλεσης του προγράμματος και επεξηγήσεις αυτών.

Παράδειγμα 1: Εκτελώντας το πρόγραμμα και δίνοντας στην γραμμή εντολών:

`./kiwi-bench read_write 5 5 4` (5 read, 5 write, 4 threads),

αρχικά εμφανίζονται πληροφορίες για το περιβάλλον του υπολογιστή και για το header. Το πρόγραμμα εκτελείται με 4 νήματα από τα οποία τα 2 αναθέτονται για write και 2 για read. Γίνονται πρώτα 5 εισαγωγές κλειδιών-τιμών και στην συνέχεια 5 αναζητήσεις κλειδιών-τιμών από τα οποία βρίσκει και τα 5.

Ο χρόνος εκτέλεσης είναι 0sec και για αυτό η ρυθμοαπόδοση είναι ίση με μηδέν (δεν ωφελεί να διαιρέσουμε τα bits με το μηδέν).

Στο τέλος εμφανίζονται πληροφορίες για την κατάσταση της βάσης (πχ τα compaction και τα merge καθώς και την κατάσταση των 7 επιπέδων του sst).

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read_write 5 5 4
Keys:      16 bytes each
Values:    1000 bytes each
Entries:    10
NumofThreads: 4
IndexSize:  0.0 MB (estimated)
DataSize:   0.0 MB (estimated)
-----
Date:       Sun Mar 28 12:54:29 2021
CPU:        1 * AMD Ryzen 7 2700X Eight-Core Processor
CPUCache:
[29901] 28 Mar 12:54:29.438 . file.c:200 Creating directory structure: testdb/si
[29901] 28 Mar 12:54:29.438 . file.c:211 -> Creating testdb
[29901] 28 Mar 12:54:29.438 . file.c:211 -> Creating testdb/si
[29901] 28 Mar 12:54:29.438 . sst.c:283 Manifest file not present
1 adding key-1
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
1 searching key-1
2 searching key-2
3 searching key-3
4 searching key-4
5 searching key-5
[29901] 28 Mar 12:54:29.439 . db.c:34 Closing database 5
[29901] 28 Mar 12:54:29.439 . sst.c:595 IN sst_merge the REFCOUNT IS at 2
[29901] 28 Mar 12:54:29.439 . sst.c:415 Sending termination message to the detached thread
[29901] 28 Mar 12:54:29.439 . sst.c:422 Waiting the merger thread
[29901] 28 Mar 12:54:29.439 . sst.c:165 The merge thread received a MERGE job
[29901] 28 Mar 12:54:29.439 . sst.c:166 Merging inside compaction thread
[29901] 28 Mar 12:54:29.439 . sst.c:608 Compacting the memtable to a SST file
[29901] 28 Mar 12:54:29.439 . sst.c:879 Range [key-1, key-5] DOES NOT overlap in level 0. Checking others
[29901] 28 Mar 12:54:29.439 . sst.c:825 Extracted range: [key-1, key-5]
[29901] 28 Mar 12:54:29.439 . sst.c:825 Extracted range: [key-1, key-5]
[29901] 28 Mar 12:54:29.439 . sst.c:931 Using level 2 for memtable compaction [key-1, key-5]
[29901] 28 Mar 12:54:29.440 . file.c:200 Creating directory structure: testdb/si/2
[29901] 28 Mar 12:54:29.439 . file.c:211 -> Creating testdb/si/2
[29901] 28 Mar 12:54:29.439 . sst.c:633 Compaction of 5 [5095 bytes allocated] elements started
[29901] 28 Mar 12:54:29.439 . sst_builder.c:167 Index block @ offset: 0x199 size: 30
[29901] 28 Mar 12:54:29.439 . sst_builder.c:168 Meta block @ offset: 0x151 size: 72
[29901] 28 Mar 12:54:29.440 . sst_builder.c:171 Bloom block @ offset: 0x141 size: 16
[29901] 28 Mar 12:54:29.440 . file.c:170 Truncating file testdb/si/2/0.sst to 511 bytes
[29901] 28 Mar 12:54:29.441 . file.c:65 Mapping of 511 bytes for testdb/si/2/0.sst
[29901] 28 Mar 12:54:29.441 . sst_loader.c:183 Index @ offset: 409 size: 30
[29901] 28 Mar 12:54:29.441 . sst_loader.c:184 Meta Block @ offset: 337 size: 72
[29901] 28 Mar 12:54:29.441 . sst_loader.c:201 Data size: 321
[29901] 28 Mar 12:54:29.441 . sst_loader.c:203 Index size: 0
[29901] 28 Mar 12:54:29.441 . sst_loader.c:204 Key size: 80
[29901] 28 Mar 12:54:29.441 . sst_loader.c:205 Num blocks size: 1
[29901] 28 Mar 12:54:29.441 . sst_loader.c:206 Num entries size: 5
[29901] 28 Mar 12:54:29.441 . sst_loader.c:207 Value size: 5000
[29901] 28 Mar 12:54:29.441 . sst_loader.c:210 Filter size: 16
[29901] 28 Mar 12:54:29.441 . sst_loader.c:211 Bloom offset 321 size: 16
[29901] 28 Mar 12:54:29.441 . sst.c:635 Compaction of 5 elements finished
[29901] 28 Mar 12:54:29.441 . file.c:170 Truncating file testdb/si/manifest to 44 bytes
[29901] 28 Mar 12:54:29.442 . sst.c:51 --- Level 0 [ 0 files, 0 bytes]---
[29901] 28 Mar 12:54:29.442 . sst.c:51 --- Level 1 [ 0 files, 0 bytes]---
[29901] 28 Mar 12:54:29.442 . sst.c:51 --- Level 2 [ 1 files, 511 bytes]---
[29901] 28 Mar 12:54:29.442 . sst.c:60 Metadata fillenum:0 smallest: key-1 largest: key-5
[29901] 28 Mar 12:54:29.442 . sst.c:51 --- Level 3 [ 0 files, 0 bytes]---
[29901] 28 Mar 12:54:29.442 . sst.c:51 --- Level 4 [ 0 files, 0 bytes]---
[29901] 28 Mar 12:54:29.442 . sst.c:51 --- Level 5 [ 0 files, 0 bytes]---
[29901] 28 Mar 12:54:29.442 . sst.c:51 --- Level 6 [ 0 files, 0 bytes]---
[29901] 28 Mar 12:54:29.442 . log.c:46 Removing old log file testdb/si/-1.log
[29901] 28 Mar 12:54:29.442 . sst.c:170 Merge successfully completed. Releasing the skiplist
[29901] 28 Mar 12:54:29.442 . skiplist.c:57 Skiplist refcount is at 0. Freeing up the structure
[29901] 28 Mar 12:54:29.442 . sst.c:176 Exiting from the merge thread as user requested
[29901] 28 Mar 12:54:29.442 . file.c:170 Truncating file testdb/si/manifest to 44 bytes
[29901] 28 Mar 12:54:29.442 . log.c:46 Removing old log file testdb/si/0.log
-----
[Random-Read Write ThroughPut: 0 (bit/sec); Total cost:0.000(sec)
[Random-Write (done:5): 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec)
[Random-Read (found:5): 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

Εικόνα 1. Εκτέλεση `./kiwi-bench read_write 5 5 4`

Παράδειγμα 2: Εκτελώντας το πρόγραμμα και δίνοντας στην γραμμή εντολών:

```
./kiwi-bench read_write 102 100 7 (102 read, 100 write, 7 threads)
```

Εκτελούνται σωστά οι λειτουργίες (102)read και (100)write. Από τις 102 αναζητήσεις κλειδιών-τιμών βρίσκει μόνο τις 100 όπως φαίνεται στην Εικόνα 3.

Στο τέλος της εκτέλεσης εμφανίζεται ο αριθμός των αιτήσεων που έγιναν για read και write καθώς και τον χρόνο εκτέλεσης που είναι ίσος με μηδέν. Γι' αυτόν τον λόγο η ρυθμοαπόδοση είναι ίση με μηδέν.

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read_write 102 100 7
Keys:          16 bytes each
Values:        1000 bytes each
Entries:       202
NumofThreads:  7
IndexSize:     0.0 MB (estimated)
DataSize:      0.2 MB (estimated)
-----
Date:          Sun Mar 28 12:55:20 2021
CPU:           1 * AMD Ryzen 7 2700X Eight-Core Processor
CPU/Cache:
[29920] 28 Mar 12:55:20.356 . file.c:200 Creating directory structure: testdb/si
[29920] 28 Mar 12:55:20.356 - file.c:211 -> Creating testdb
[29920] 28 Mar 12:55:20.356 - file.c:211 -> Creating testdb/si
[29920] 28 Mar 12:55:20.356 . sst.c:283 Manifest file not present
1 adding key-1
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
```

Εικόνα 2. Εκτέλεση ./kiwi-bench read_write 102 100 7

```
95 searching key-95
96 searching key-96
97 searching key-97
98 searching key-98
99 searching key-99
100 searching key-100
101 searching key-101
[29920] 28 Mar 12:55:20.360 . kiwi.c:174 not found key#key-101
102 searching key-102
[29920] 28 Mar 12:55:20.360 . kiwi.c:174 not found key#key-102
[29920] 28 Mar 12:55:20.360 . db.c:34 Closing database 100
```

Εικόνα 3. Εμφάνιση κλειδιών 101 και 102

```
[29920] 28 Mar 12:55:20.362 . sst_loader.c:201 Data size: 6377
[29920] 28 Mar 12:55:20.362 . sst_loader.c:203 Index size: 0
[29920] 28 Mar 12:55:20.362 . sst_loader.c:204 Key size: 1600
[29920] 28 Mar 12:55:20.362 . sst_loader.c:205 Num blocks size: 20
[29920] 28 Mar 12:55:20.362 . sst_loader.c:206 Num entries size: 100
[29920] 28 Mar 12:55:20.362 . sst_loader.c:207 Value size: 100000
[29920] 28 Mar 12:55:20.362 . sst_loader.c:210 Filter size: 244
[29920] 28 Mar 12:55:20.362 . sst_loader.c:211 Bloom offset 6377 size: 244
[29920] 28 Mar 12:55:20.362 . sst.c:635 Compaction of 100 elements finished
[29920] 28 Mar 12:55:20.362 - file.c:170 Truncating file testdb/si/manifest to 44 bytes
[29920] 28 Mar 12:55:20.362 . sst.c:51 --- Level 0 [ 0 files, 0 bytes]---
[29920] 28 Mar 12:55:20.362 . sst.c:51 --- Level 1 [ 0 files, 0 bytes]---
[29920] 28 Mar 12:55:20.362 . sst.c:51 --- Level 2 [ 1 files, 7 KiB ]---
[29920] 28 Mar 12:55:20.362 . sst.c:60 Metadata filename:0 smallest: key-1 largest: key-99
[29920] 28 Mar 12:55:20.362 . sst.c:51 --- Level 3 [ 0 files, 0 bytes]---
[29920] 28 Mar 12:55:20.362 . sst.c:51 --- Level 4 [ 0 files, 0 bytes]---
[29920] 28 Mar 12:55:20.362 . sst.c:51 --- Level 5 [ 0 files, 0 bytes]---
[29920] 28 Mar 12:55:20.362 . sst.c:51 --- Level 6 [ 0 files, 0 bytes]---
[29920] 28 Mar 12:55:20.362 . log.c:46 Removing old log file testdb/si/-1.log
[29920] 28 Mar 12:55:20.362 . sst.c:170 Merge successfully completed. Releasing the skiplist
[29920] 28 Mar 12:55:20.362 . skiplist.c:57 SkipList refcount is at 0. Freeing up the structure
[29920] 28 Mar 12:55:20.363 - sst.c:176 Exiting from the merge thread as user requested
[29920] 28 Mar 12:55:20.363 - file.c:170 Truncating file testdb/si/manifest to 44 bytes
[29920] 28 Mar 12:55:20.363 . log.c:46 Removing old log file testdb/si/0.log
+-----+
|Random-Read Write ThroughPut: 0 (bit/sec); Total cost:0.000(sec)
|Random-Write (done:100): 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec)
|Random-Read (found:100): 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

Εικόνα 4. Εμφάνιση του τέλους του προγράμματος

Παράδειγμα 3: Εκτελώντας το πρόγραμμα και δίνοντας στην γραμμή εντολών:

`./kiwi-bench read_write 69999 70000 16` (69999 read και 70000 write με 16 threads)

εκτελούνται 70000 εγγραφές κλειδιών-τιμών και 69999 αναζητήσεις κλειδιών-τιμών από τα οποία βρίσκει και τα 69999 (Εικόνα 5-6).

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read_write 69999 70000 16
Keys:          16 bytes each
Values:        1000 bytes each
Entries:       139999
NumofThreads:  16
IndexSize:     3.3 MB (estimated)
DataSize:      134.0 MB (estimated)
-----
Date:          Sun Mar 28 12:56:02 2021
CPU:           1 * AMD Ryzen 7 2700X Eight-Core Processor
CPUcache:
[29941] 28 Mar 12:56:02.560 . file.c:200 Creating directory structure: testdb/si
[29941] 28 Mar 12:56:02.560 - file.c:211 -> Creating testdb
[29941] 28 Mar 12:56:02.561 - file.c:211 -> Creating testdb/si
[29941] 28 Mar 12:56:02.561 . sst.c:283 Manifest file not present
1 adding key-1
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
6 adding key-6
7 adding key-7
8 adding key-8
```

Εικόνα 5. Εκτέλεση `./kiwi-bench read_write 69999 70000 16`

```
[29941] 28 Mar 12:56:03.671 - sst.c:176 Exiting from the merge thread as user requested
[29941] 28 Mar 12:56:03.672 - file.c:170 Truncating file testdb/si/manifest to 116 bytes
[29941] 28 Mar 12:56:03.675 . log.c:46 Removing old log file testdb/si/17.log
-----+-----
|Random-Read Write ThroughPut: 4479968 (bit/sec); Total cost:1.000(sec)
|Random-Write (done:70000): 0.000014 sec/op; 70000.0 writes/sec(estimated); cost:1.000(sec)
|Random-Read (found:69999): 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

Εικόνα 6. Εμφάνιση του τέλους εκτέλεσης του προγράμματος

ΣΤΑΤΙΣΤΙΚΑ

Για την εξαγωγή στατιστικών, θα υλοποιήσουμε την εκτέλεση του προγράμματος απαιτώντας 600.000 reads και 600.000 writes με τον αριθμό των νημάτων να είναι αρχικά 2 και στην συνέχεια 4,8,16,32,64,128.

Εκτέλεση του προγράμματος με 2 νήματα:

Αρχικά εκτελώντας `./kiwi-bench read_write 600000 600000 2`

το πρόγραμμα δημιουργεί 600000 εγγραφές κλειδιών-τιμών και στην συνέχεια κάνει 600000 αναζητήσεις κλειδιών-τιμών όπου και τις βρίσκει όλες (όπως φαίνεται και στην Εικόνα 7-8).

Ο χρόνος απόκρισης είναι 19sec εκ των οποίων τα 11sec αφορούν τις εγγραφές και τα 8sec την αναζήτηση (Εικόνα 8).

Επίσης, η ρυθμοαπόδοση είναι ίση με 2021052 bit/sec.


```

myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read_write 600000 600000 2
Keys:          16 bytes each
Values:        1000 bytes each
Entries:        1200000
NumofThreads:   2
IndexSize:     28.6 MB (estimated)
DataSize:      1149.0 MB (estimated)
-----
Date:          Sun Mar 28 13:04:25 2021
CPU:           1 * AMD Ryzen 7 2700X Eight-Core Processor
CPUCache:
[29982] 28 Mar 13:04:25.880 . file.c:200 Creating directory structure: testdb/si
[29982] 28 Mar 13:04:25.880 - file.c:211 -> Creating testdb
[29982] 28 Mar 13:04:25.880 - file.c:211 -> Creating testdb/si
[29982] 28 Mar 13:04:25.880 . sst.c:283 Manifest file not present
1 adding key-1
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
6 adding key-6
7 adding key-7

```

Εικόνα 7. Εκτέλεση ./kiwi-bench read_write 600000 600000 2

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Random-Read_Write ThroughPut: 2021052 (bit/sec); Total cost:19.000(sec)
|Random-Write   (done:600000):  0.000018 sec/op; 54545.5 writes/sec(estimated); cost:11.000(sec)
|Random-Read    (found:600000):  0.000013 sec/op; 75000.0 reads/sec(estimated); cost:8.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$

```

Εικόνα 8. Εμφάνιση του τέλους εκτέλεσης του προγράμματος

Εκτέλεση του προγράμματος με 4 νήματα:

Εκτελώντας το πρόγραμμα με 4 νήματα, το πρόγραμμα εκτελείται γρηγορότερα αφού ο χρόνος απόκρισης είναι ίσος με τα 17sec, 10 για τις εγγραφές και 7 για τις αναζητήσεις.

Ακόμη, η ρυθμοαπόδοση είναι ίση με 2258823 bit/sec, όπως είναι αναμενόμενο αφού αυξάνοντας τα νήματα περιμένουμε ότι ο όγκος των bits θα εγγράφεται και θα αναζητείται γρηγορότερα από ό,τι με 2 νήματα.

```

myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read_write 600000 600000 4
Keys:          16 bytes each
Values:        1000 bytes each
Entries:        1200000
NumofThreads:   4
IndexSize:     28.6 MB (estimated)
DataSize:      1149.0 MB (estimated)
-----
Date:          Sun Mar 28 13:08:00 2021
CPU:           1 * AMD Ryzen 7 2700X Eight-Core Processor
CPUCache:
[30108] 28 Mar 13:08:00.078 . file.c:200 Creating directory structure: testdb/si
[30108] 28 Mar 13:08:00.078 - file.c:211 -> Creating testdb
[30108] 28 Mar 13:08:00.078 - file.c:211 -> Creating testdb/si
[30108] 28 Mar 13:08:00.078 . sst.c:283 Manifest file not present
1 adding key-1
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
6 adding key-6
7 adding key-7

```

Εικόνα 9. Εκτέλεση ./kiwi-bench read_write 600000 600000 4


```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Random-Read Write ThroughPut: 2258823 (bit/sec); Total cost:17.000(sec)
|Random-Write   (done:600000):  0.000017 sec/op; 60000.0 writes/sec(estimated); cost:10.000(sec)
|Random-Read    (found:600000):  0.000012 sec/op; 85714.3 reads/sec(estimated); cost:7.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$

```

Εικόνα 10. Εμφάνιση του τέλους εκτέλεσης του προγράμματος

Εκτέλεση του προγράμματος με 8 νήματα:

Εκτελώντας το πρόγραμμα με 8 νήματα, το πρόγραμμα εκτελείται γρηγορότερα αφού ο χρόνος απόκρισης είναι ίσος με τα 12sec, 7 για τις εγγραφές και 5 για τις αναζητήσεις.

Ακόμη, η ρυθμοαπόδοση είναι ίση με 3200000 bit/sec, όπως είναι αναμενόμενο αφού αυξάνοντας τα νήματα περιμένουμε ότι ο όγκος των bits θα εγγράφεται και θα αναζητείται γρηγορότερα από ό,τι με λιγότερα νήματα.

```

myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read_write 600000 600000 8
Keys:          16 bytes each
Values:        1000 bytes each
Entries:       1200000
NumofThreads:  8
IndexSize:     28.6 MB (estimated)
DataSize:      1149.0 MB (estimated)
-----
Date:          Sun Mar 28 13:09:25 2021
CPU:           1 * AMD Ryzen 7 2700X Eight-Core Processor
CPUCache:
[30145] 28 Mar 13:09:25.479 . file.c:200 Creating directory structure: testdb/si
[30145] 28 Mar 13:09:25.479 - file.c:211 -> Creating testdb
[30145] 28 Mar 13:09:25.479 - file.c:211 -> Creating testdb/si
[30145] 28 Mar 13:09:25.479 . sst.c:283 Manifest file not present
1 adding key-1
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
6 adding key-6
7 adding key-7
8 adding key-8
9 adding key-9
10 adding key-10
11 adding key-11

```

Εικόνα 11. Εκτέλεση ./kiwi-bench read_write 600000 600000 8

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Random-Read Write ThroughPut: 3200000 (bit/sec); Total cost:12.000(sec)
|Random-Write   (done:600000):  0.000012 sec/op; 85714.3 writes/sec(estimated); cost:7.000(sec)
|Random-Read    (found:600000):  0.000008 sec/op; 120000.0 reads/sec(estimated); cost:5.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$

```

Εικόνα 12. Εμφάνιση του τέλους εκτέλεσης του προγράμματος

Εκτέλεση του προγράμματος με 16 νήματα:

Εκτελώντας το πρόγραμμα με 16 νήματα (Εικόνα 13), το πρόγραμμα εκτελείται γρηγορότερα αφού ο χρόνος απόκρισης είναι ίσος με τα 11sec, 7 για τις εγγραφές και 4 για τις αναζητήσεις (Εικόνα 14).

Ακόμη, η ρυθμοαπόδοση είναι ίση με 3490989 bit/sec, όπως είναι αναμενόμενο αφού αυξάνοντας τα νήματα περιμένουμε ότι ο όγκος των bits θα εγγράφεται και θα αναζητείται γρηγορότερα από ό,τι με λιγότερα νήματα.

```

myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read_write 600000 600000 16
Keys:          16 bytes each
Values:        1000 bytes each
Entries:       1200000
NumofThreads:  16
IndexSize:     28.6 MB (estimated)
DataSize:      1149.0 MB (estimated)
-----
Date:          Sun Mar 28 13:12:58 2021
CPU:           1 * AMD Ryzen 7 2700X Eight-Core Processor
CPUCache:
[30362] 28 Mar 13:12:58.446 . file.c:200 Creating directory structure: testdb/si
[30362] 28 Mar 13:12:58.446 - file.c:211 -> Creating testdb
[30362] 28 Mar 13:12:58.446 - file.c:211 -> Creating testdb/si
[30362] 28 Mar 13:12:58.446 . sst.c:283 Manifest file not present
1 adding key-1
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
6 adding key-6
7 adding key-7
8 adding key-8
9 adding key-9
10 adding key-10

```

Εικόνα 13. Εκτέλεση ./kiwi-bench read_write 600000 600000 16

```

+-----+
|Random-Read_Write ThroughPut: 3490909 (bit/sec); Total cost:11.000(sec)
|Random-Write   (done:600000):  0.000012 sec/op; 85714.3 writes/sec(estimated); cost:7.000(sec)
|Random-Read    (found:600000):  0.000007 sec/op; 150000.0 reads/sec(estimated); cost:4.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$

```

Εικόνα 14. Εμφάνιση του τέλους εκτέλεσης του προγράμματος

Εκτέλεση του προγράμματος με 32 νήματα:

Εκτελώντας το πρόγραμμα με 32 νήματα (Εικόνα 15), το πρόγραμμα εκτελείται γρηγορότερα αφού ο χρόνος απόκρισης είναι ίσος με τα 11sec, 7 για τις εγγραφές και 4 για τις αναζητήσεις (Εικόνα 16).

Ακόμη, η ρυθμοαπόδοση είναι ίση με 3490989 bit/sec, όπως είναι αναμενόμενο αφού αυξάνοντας τα νήματα περιμένουμε ότι ο όγκος των bits θα εγγράφεται και θα αναζητείται γρηγορότερα από ό,τι με λιγότερα νήματα.

```

myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read_write 600000 600000 32
Keys:          16 bytes each
Values:        1000 bytes each
Entries:       1200000
NumofThreads:  32
IndexSize:     28.6 MB (estimated)
DataSize:      1149.0 MB (estimated)
-----
Date:          Sun Mar 28 13:16:42 2021
CPU:           1 * AMD Ryzen 7 2700X Eight-Core Processor
CPUCache:
[30862] 28 Mar 13:16:42.886 . file.c:200 Creating directory structure: testdb/si
[30862] 28 Mar 13:16:42.886 - file.c:211 -> Creating testdb
[30862] 28 Mar 13:16:42.886 - file.c:211 -> Creating testdb/si
[30862] 28 Mar 13:16:42.886 . sst.c:283 Manifest file not present
1 adding key-1
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
6 adding key-6
7 adding key-7
8 adding key-8
9 adding key-9

```

Εικόνα 15. Εκτέλεση ./kiwi-bench read_write 600000 600000 32

```

+-----+-----+-----+-----+-----+-----+-----+-----+
|Random-Read Write ThroughPut: 3490909 (bit/sec); Total cost:11.000(sec)
|Random-Write   (done:600000):  0.000012 sec/op; 85714.3 writes/sec(estimated); cost:7.000(sec)
|Random-Read    (found:600000):  0.000007 sec/op; 150000.0 reads/sec(estimated); cost:4.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$

```

Εικόνα 16. Εμφάνιση του τέλους εκτέλεσης του προγράμματος

Εκτέλεση του προγράμματος με 64 νήματα:

Εκτελώντας το πρόγραμμα με 64 νήματα (Εικόνα 17), το πρόγραμμα εκτελείται γρηγορότερα αφού ο χρόνος απόκρισης είναι ίσος με τα 10sec, 7 για τις εγγραφές και 3 για τις αναζητήσεις (Εικόνα 18).

Ακόμη, η ρυθμοαπόδοση είναι ίση με 3840000 bit/sec, όπως είναι αναμενόμενο αφού αυξάνοντας τα νήματα περιμένουμε ότι ο όγκος των bits θα εγγράφεται και θα αναζητείται γρηγορότερα από ό,τι με λιγότερα νήματα.

```

myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read_write 600000 600000 64
Keys:      16 bytes each
Values:    1000 bytes each
Entries:   1200000
NumofThreads: 64
IndexSize: 28.6 MB (estimated)
DataSize:  1149.0 MB (estimated)
-----
Date:      Sun Mar 28 13:14:15 2021
CPU:       1 * AMD Ryzen 7 2700X Eight-Core Processor
CPU Cache:
[30438] 28 Mar 13:14:15.686 . file.c:200 Creating directory structure: testdb/si
[30438] 28 Mar 13:14:15.686 - file.c:211 -> Creating testdb
[30438] 28 Mar 13:14:15.686 - file.c:211 -> Creating testdb/si
[30438] 28 Mar 13:14:15.686 . sst.c:283 Manifest file not present
1 adding key-1
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
6 adding key-6
7 adding key-7
8 adding key-8

```

Εικόνα 17. Εκτέλεση ./kiwi-bench read_write 600000 600000 64

```

+-----+-----+-----+-----+-----+-----+-----+-----+
|Random-Read Write ThroughPut: 3840000 (bit/sec); Total cost:10.000(sec)
|Random-Write   (done:600000):  0.000012 sec/op; 85714.3 writes/sec(estimated); cost:7.000(sec)
|Random-Read    (found:600000):  0.000005 sec/op; 200000.0 reads/sec(estimated); cost:3.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$

```

Εικόνα 18. Εμφάνιση του τέλους εκτέλεσης του προγράμματος

Εκτέλεση του προγράμματος με 128 νήματα:

Εκτελώντας το πρόγραμμα με 128 νήματα (Εικόνα 19), το πρόγραμμα εκτελείται γρηγορότερα αφού ο χρόνος απόκρισης είναι ίσος με τα 10sec, 7 για τις εγγραφές και 3 για τις αναζητήσεις (Εικόνα 20).

Ακόμη, η ρυθμοαπόδοση είναι ίση με 3840000 bit/sec, όπως είναι αναμενόμενο αφού αυξάνοντας τα νήματα περιμένουμε ότι ο όγκος των bits θα εγγράφεται και θα αναζητείται γρηγορότερα από ό,τι με λιγότερα νήματα.

```

myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read_write 600000 600000 128
Keys:          16 bytes each
Values:        1000 bytes each
Entries:       1200000
NumofThreads:  128
IndexSize:     28.6 MB (estimated)
DataSize:      1149.0 MB (estimated)
-----
Date:          Sun Mar 28 13:16:02 2021
CPU:           1 * AMD Ryzen 7 2700X Eight-Core Processor
CPUCache:
[30720] 28 Mar 13:16:02.377 . file.c:200 Creating directory structure: testdb/si
[30720] 28 Mar 13:16:02.377 - file.c:211 -> Creating testdb
[30720] 28 Mar 13:16:02.377 - file.c:211 -> Creating testdb/si
[30720] 28 Mar 13:16:02.377 . sst.c:283 Manifest file not present
1 adding key-1
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
6 adding key-6
7 adding key-7
8 adding key-8
9 adding key-9
10 adding key-10

```

Εικόνα 19. Εκτέλεση ./kiwi-bench read_write 600000 600000 128

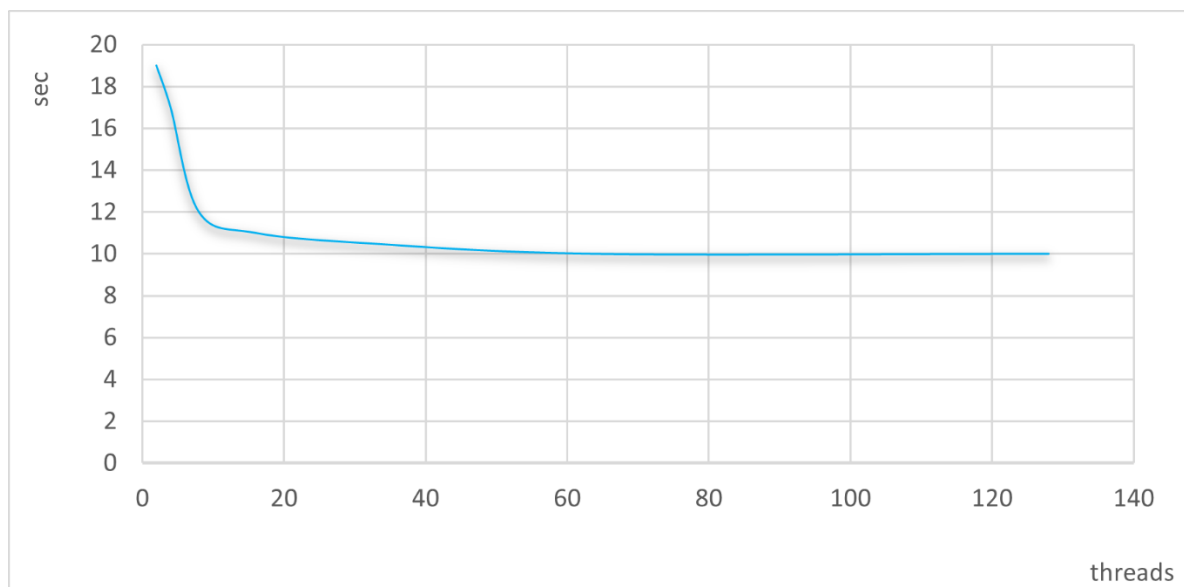
```

+-----+
|Random-Read Write ThroughPut: 3840000 (bit/sec); Total cost:10.000(sec)
|Random-Write   (done:600000):  0.000012 sec/op; 85714.3 writes/sec(estimated); cost:7.000(sec)
|Random-Read    (found:600000):  0.000005 sec/op; 200000.0 reads/sec(estimated); cost:3.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$

```

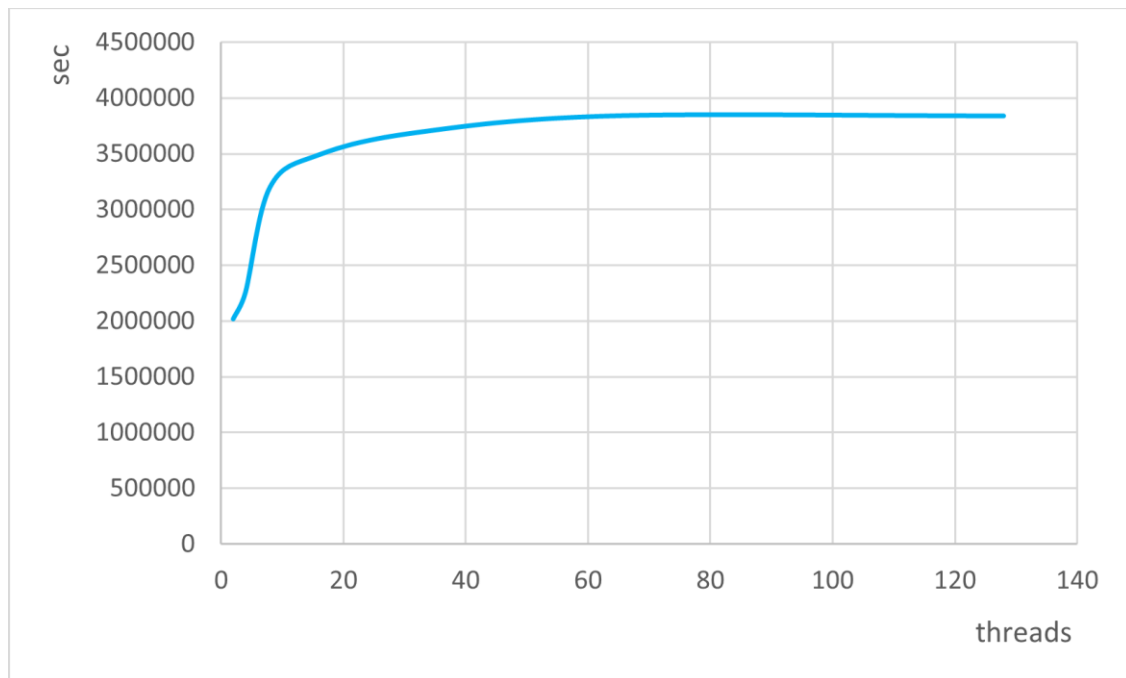
Εικόνα 20. Εμφάνιση του τέλους εκτέλεσης του προγράμματος

Τα παραπάνω αποτελέσματα τα περνάμε σε excel και βγάζουμε τις παρακάτω γραφικές παραστάσεις.



Εικόνα 21. Γραφική παράσταση για την απόδοση της read_write

Όπως φαίνεται στην (Εικόνα 21), η απόδοση αυξάνεται παροδικά του χρόνου με την αύξηση των νημάτων. Για περισσότερα νήματα, ο χρόνος απόδοσης μειώνεται.



Εικόνα 21. Γραφική παράσταση για την απόδοση με βάση της ρυθμοαπόδοσης

Όπως φαίνεται από τα παραπάνω στατιστικά, με την αύξηση των νημάτων, το βέλτιστο κόστος είναι 10sec με 64 νήματα, και μετά αυξάνοντας τα νήματα δεν υπάρχει άλλη βελτίωση.

➤ *Όρια εκτέλεσης του προγράμματος:*

Για λιγότερα από 100.000 read&write με 2 νήματα, το κόστος θα είναι μηδέν.

Για περισσότερα από 4 εκατομμύρια read&write με 2 νήματα, θα δημιουργηθεί bus error, όπως είναι αναμενόμενο αφού η βάση έχει όριο χωρητικότητας.