Brendan Murphy

Problem Set 4

Part 1

Problem 1: Replication

Replication improves performance by placing records used most frequently close to the client accessing them. Replication also provides high availability by remaining available if there is a failure at one site. Some challenges include determining the best way to execute a query because of factoring in communication costs, and ensuring data consistency by having data held in multiple places.

Context: System uses voting and fully distributed locking

To write; transaction needs $x > n/2$

To read; transaction needs $s > n - x$

$n = 7$

1. Update 6 copies, read 3 copies

   $x = 6$, $s = 3$ This voting scheme will not work.

   $6 > 7 / 2$ and **$3! > 7 - 6$**

2. Update 2 copies, read 6 copies

   $x = 2$, $s = 6$ This voting scheme will not work.

   $2! > 7 / 2$ and **$6 > 7 - 2$**

3. Update 5 copies, read 2 copies

   $x = 5$, $s = 2$ This voting scheme will not work.

   $5 > 7 / 2$ and **$2! > 7 - 5$**

4. Update 4 copies, read 4 copies

   $x = 4$, $s = 4$ This voting scheme will work.

   $4 > 7 / 2$ and **$4 > 7 - 4$**

Problem 2: Distributed locking with update locks

To acquire a global exclusive lock, acquire local exclusive locks for a sufficient number of copies. For this to work we need (x > n/2) where n is the total number of copies and x is the number of copies that must be locked to acquire a global exclusive lock. In other words, the transaction needs more than half to acquire the lock.

To acquire a global shared lock, acquire local shared locks for a sufficient number of copies. For this to work the number of local shared locks acquired must be greater that the total number of copies minus the number locked for the exclusive lock: (s > n -x) If there is a global exclusive lock on an item, there won`t be enough unlocked copes for a global shared lock.

An update lock allows transactions to read an item and can be upgraded to an exclusive lock. To acquire a global update lock the transaction must acquire a at least as many local copies of exclusive locks in order to satisfy the restraint that there can only be 1 update lock at a time (u >= x).

Problem 3: Object-oriented data models

1. ODL schema definition:

```
class Author (key id) {
        attribute string id;
        Struct name{ string first_name, string middle_name, string last_name }
        attribute date dob;

        relationship Set<Book> booksWritten
                inverse Book::author;
}

class Book (key isbn) {
        attribute string isbn;
        attribute string title;
        attribute string publisher;
        attribute int num_pages;
        attribute string genre;

        relationship Set <Author> authors
                inverse Author::booksWritten;
}
```

2: Object-relational schema definition:

Author (id, name (first_name, middle_name, last_name), dob, booksWritten ({*Book}));

Book (isbn, title, publisher, num_pages, genre, writtenBy ({*Author}));

Part 3: Queries

Query 1

```
SELECT Count (*)
FROM Book
WHERE Book.WrittenBy ->last_name = 'Sullivan'
GROUP BY Book.isbn;
```

Query 2

```
SELECT Book.title
FROM Book, unnest(Book.authors) As A(Author)
WHERE A.author->first_name = 'J.K.'
AND A.author->last_name = 'Rowling'
AND Book.publisher LIKE 'Hogwarts%';
```