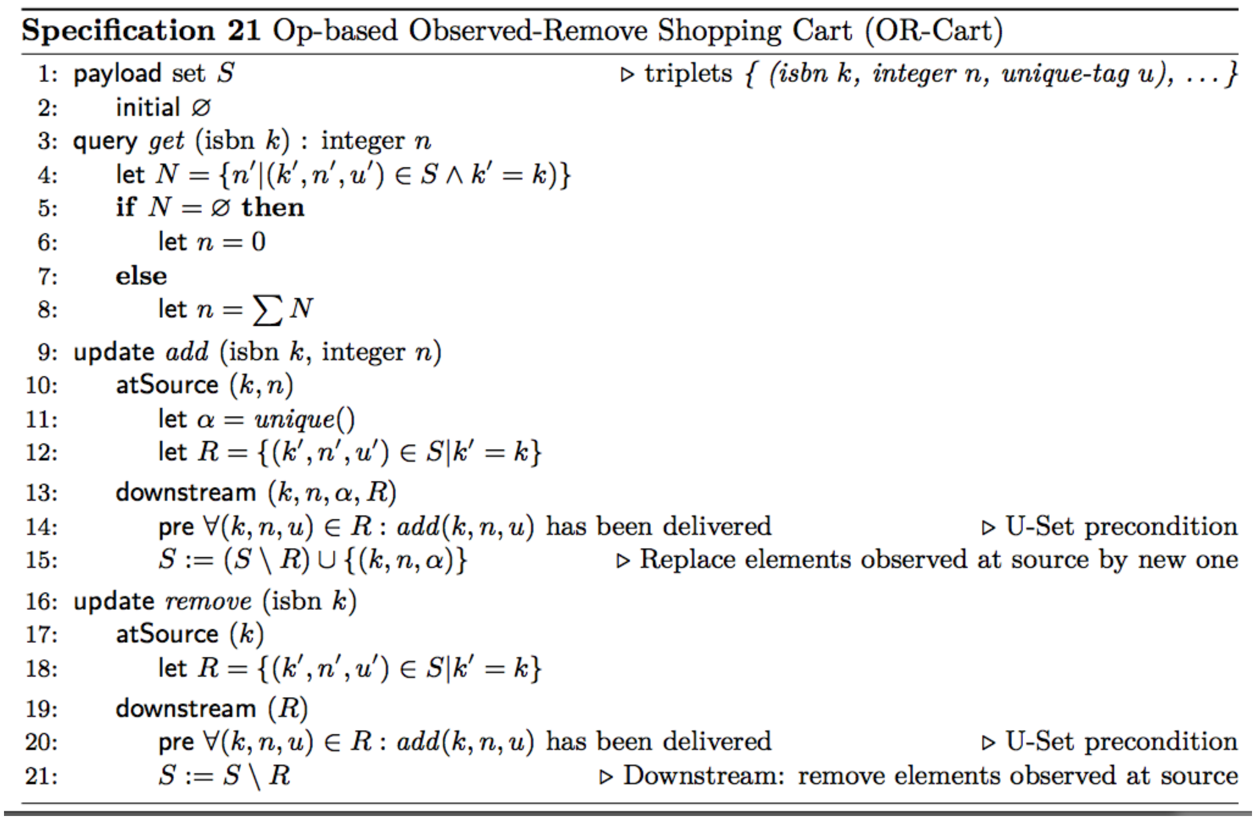
Project Proposal Draft

Ever since the CAP theorem was conjectured, practitioners of distributed systems in academia and industry have evaluated various ways of making trade-offs between consistency, availability, and network partition tolerance to strive for the best possible user experience. Internet scale data storage systems often have to aim for the high availability to serve a large number of concurrent requests while being resilient to network partitioning caused by failure of various hardware/software components. Amazon's highly available Dynamo key-value store is a prime example of such a distributed system. Amazon engineers chose the eventually consistent model to provide a reliable "always on" experience to the users. The sacrifices in data consistency will manifest under certain usage scenarios. For instance, in their 2007 publication, the authors mentioned an anomaly in the Amazon shopping cart in which deleted items can reappear under certain conditions [1]. Although these anomalies are very rare and can always be corrected by the user, customer experience can be improved by minimizing these kind of anomalies.

Distributed systems use optimistic replication to achieve better availability and performance. Updates to data can be made at some replica without synchronization with other replica. Conflicting updates at different replica are usually resolved through a background consensus algorithm and some updates may need to be rolled-back. The downslide to this approach is that conflict resolution mechanism can be complex and error-prone. In the distributed systems literature, the concept of Conflict-free Replicated Data Type (CRDT) has been proposed to help with this problem. CRDT is a theoretical construct that ensures eventual consistency by design without the use of consensus. CRDT presents a theoretical-sound approach to eventual consistency that promise to remove the complexity in conflict resolution in existing implementations. However, the level of consistency can be be weak without the use of consensus. More specifically, CRDT will not provide sequential consistency which is often desired by application developers.

We propose to implement a proof of concept E-commerce shopping cart using CRDT. In their 2011 technical report, Shapiro and co-authors presented a comprehensive review of literature on CRDT and outlined the idea of using an Observed-Remove Set (OR-Set) to implement E-commerce shopping cart. The OR-Set is an example of what the authors refer to as the operation-based Commutative Replicated Data Type. In these type of constructs, operations at a given replica need to commute with each other. In simple terms, two operations f(r) and g(r) are said to commute with each other if the order of the two operations does not affect the final state at the replica (i.e. $g(f(r)) = f(g(r))$). A simple set data type we learned from basic algorithm class, which supports add and remove operation, is not a CRDT. This is because addition after removal vs. removal after addition yield different final states for the set. One way to convert the simple set into a CRDT is attaching a unique identifier to each element of the set and ensure addition of an element is always delivered before removal of the same element. This extension to the simple set is referred to as the U-Set by the authors. The authors proposed to implement the Observed-Remove shopping cart (OR-cart) by extending the U-Set into a U-Map.



Specification 21 Op-based Observed-Remove Shopping Cart (OR-Cart)

```
payload set $S$                                          ▷ triplets { (isbn $k$, integer $n$, unique-tag $u$), ... }
    initial $\varnothing$
query get (isbn $k$) : integer $n$
    let $N = \{n' | (k', n', u') \in S \wedge k' = k)\}$
    if $N = \varnothing$ then
        let $n = 0$
    else
        let $n = \sum N$
update add (isbn $k$, integer $n$)
    atSource $(k, n)$
        let $\alpha = unique()$
        let $R = \{(k', n', u') \in S | k' = k\}$
    downstream $(k, n, \alpha, R)$
        pre $\forall (k, n, u) \in R : add(k, n, u)$ has been delivered      ▷ U-Set precondition
        $S := (S \setminus R) \cup \{(k, n, \alpha)\}$              ▷ Replace elements observed at source by new one
update remove (isbn $k$)
    atSource $(k)$
        let $R = \{(k', n', u') \in S | k' = k\}$
    downstream $(R)$
        pre $\forall (k, n, u) \in R : add(k, n, u)$ has been delivered      ▷ U-Set precondition
        $S := S \setminus R$                         ▷ Downstream: remove elements observed at source
```

Tasks: (IN PROGRESS)

1. Translate the theoretical specification into an internal protocol between nodes and write the shopping cart API.
2. Divide the implementation work between the members of the group.

SWOT Analysis (IN PROGRESS)

Strengths	Weaknesses	Opportunities	Threats
• All team members are proficient in golang	• Assignments, exams, other projects, and other commitments can hinder the completion of this system •	• Implement a system to resolve a current issue and improve user experience •	• Tight time constraints. • The paper is highly theoretical and may take a great deal of time understand and work out a practical implementation plan. • Golang is a relatively new language, future changes may cause our system to become deprecated