Algoritmul de listă optimistă a fost optimizat prin introducerea unui număr de versiune, astfel încât etapa de validare să nu necesite o iterare suplimentară prin listă pentru a verifica dacă un alt element a fost deja adăugat/șters din listă. Numărul de versionare va ține evidența numărului de operații efectuate până în momentul curent, deci având numărul de versiune putem verifica la orice moment din execuție dacă există vreo altă execuție paralelă care a realizat deja modificări asupra listei.

Pentru implementarea ideii de mai sus am introdus 2 membrii noi pentru clasa OptimisticList: version (de tip int) și versionLock (un lacăt de tip ReentrantLock). Lacătul este necesar pentru protejarea secțiunilor critice în care se fac modificări asupra variabilei version.

Astfel, funcția validate se va rezuma doar la verificarea dacă numărul de versiune al metodei curente este egal cu numărul de versiune al listei (ca și în varianta de implementare anterioară, este posibil ca un alt thread care apelează metodă să finalizeze execuția după ce unul dintre thread-uri execută cel de-al doilea while intern: while (current.key < key).

Am testat cele 2 implementări pe un program cu 4 Threaduri, care întâi adaugă în listă elemente de la 1 la 100000 și apoi șterg numerele pare de la 1 la 25000 și am obținut următorii timpi de execuție pentru 10 rulări independente:

Execution Time	OptimisticList	OptimisticListVersioning
1	60.404s	14.87s
2	58.674s	15.603s
3	61.602s	15.293s
4	59.861s	12.934s
5	57.152s	13.796s
6	56.942s	13.199s
7	56.105s	13.158s
8	56.571s	15.193s
9	53.067s	16.725s
10	53.095s	14.55s
AVG	57.82s	14.53s

Ambele implementări rulează corect și afișează rezultatul așteptat. Se observă că programul OptimisticList este îmbunătățit semnificativ din punct de vedere al timpului de execuție, mai ales în cazul unei liste cu multe numere care sunt adăugate în ordine consecutivă (la fiecare operație de add/remove/contains se va face parcurgerea integrală a listei cel puțin o dată în cazul cel mai favorabil).