Algorithm 1 SCHEDULING ALGORITHM (NEW)

Require: $\Theta = \{D_1, \cdots, D_m\}$ the set of data managed by the scheduler
Require: $\Delta_k = \{D_1^k, \cdots, D_n^k\}$ the data cache managed by the reservoir host k
Require: $\Omega(D_i) = \{k, \cdots\}$ the set of reservoir host owning data D_i
Ensure: $\Psi_k = \{D_1^k, \cdots, D_o^k\}$ the new dataset managed by the reservoir host k

1. $\Psi_k \leftarrow 0$
2. {Step 1: Remove obsolete data from cache}
3. **for all** $D_i^k \in \Delta_k$ **do**
4. **if** $((D_i^k \in \Theta) \wedge (D_i^k.lifetime.absolute > now()) \wedge (D_i^k.lifetime.relative \in \Theta))$ **then**
5. $\Psi_k \leftarrow \Psi_k \cup \{D_i^k\}$
6. **if** $(D_i^k.faultTolerant == true)$ **then**
7. update $\Omega(D_i^k)$
8. **end if**
9. **end if**
10. **end for**
11. {Step 2: Add new data to the cache}
12. **for all** $D_j \in (\Theta \setminus \Delta_k)$ **do**
13. **if** $(D_j.faultTolerant == true)$ **then**
14. update D_j
15. **end if**
16. **Let** $\alpha_k = \{D_i^k \in \Delta_k \mid D_i^k.attr = D_j.attr\}$
17. **Let** $addElement = false$
18. {Resolve affinity dependence}
19. **for all** $D_i^k \in \Delta_k$ **do**
20. **if** $((D_j.affinity == D_i^k) \wedge (D_j \notin \Delta_k))$ **then**
21. $addElement = true$
22. **end if**
23. **end for**
24. {Schedule replica}
25. **if** $((D_j.replica == -1) \bigvee (D_j.replica < |\Omega(D_j)|))$ **then**
26. $addElement = true$
27. **end if**
28. {Evaluate distrib}
29. **if** $(addElement == true)$ **then**
30. **if** $((D_j.distrib == -1) \bigvee (|\alpha_k| < D_j.distrib))$ **then**
31. $\Psi_k \leftarrow \Psi_k \cup \{D_j\}$
32. $\Omega(D_j) \leftarrow \Omega(D_j) \cup \{k\}$
33. **end if**
34. **end if**
35. **if** $(|\Psi_k \setminus \Delta_k| \geq MaxDataToSchedule)$ **then**
36. break
37. **end if**
38. **end for**
39.
40. **return** Ψ_k