ISCAS 中国科学院软件研究所学术年会'2022
暨计算机科学国家重点实验室开放周

学术论文

LPW: An Efficient Data-Aware Cache Replacement Strategy for Apache Spark
LPW: 一种面向Spark的高效缓存替换优化策略

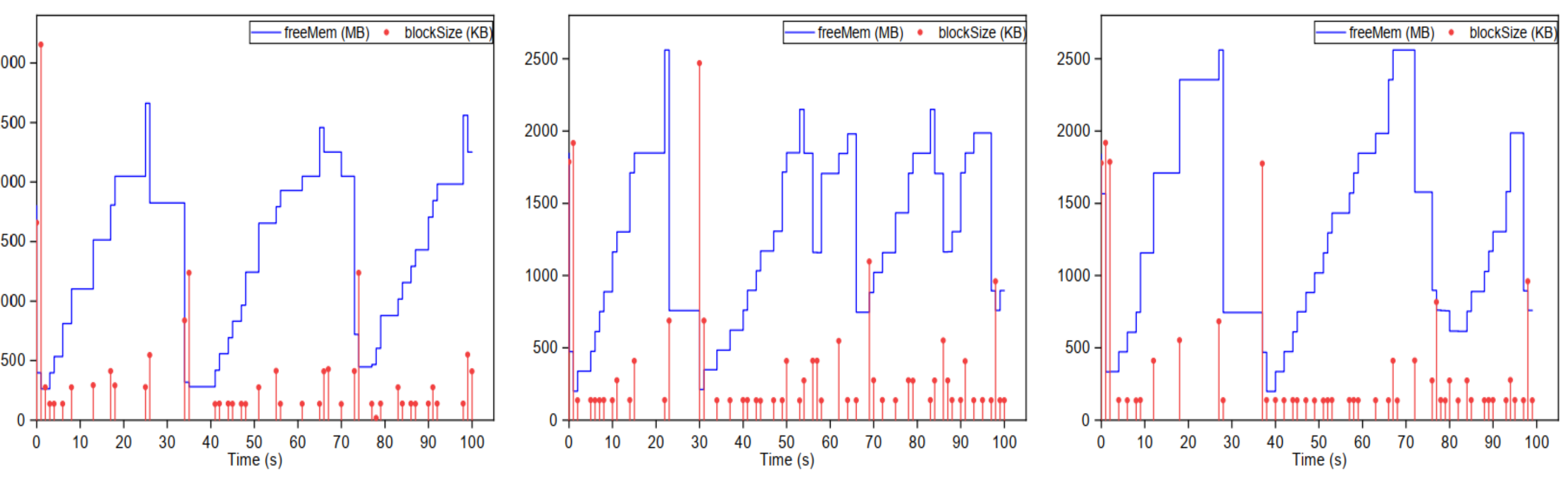
Hui Li, Shuping Ji, Hua Zhong, Wei Wang, Lijie Xu,
Zhen Tang, Jun Wei, Tao Huang

SCIENCE CHINA Information Sciences （SCIS 2022）
DOI:10.1007/s11432-021-3406-5

联系方式：李慧 (15210848139， lihui2012@otcaix.iscas.ac.cn)

Background & Motivation

Spark users usually try to cache data in memory for re-use to speed up application execution. In real-world, since the storage memory is often not enough to cache all intermediate computing results, frequent cache replacement may happen according to LRU.



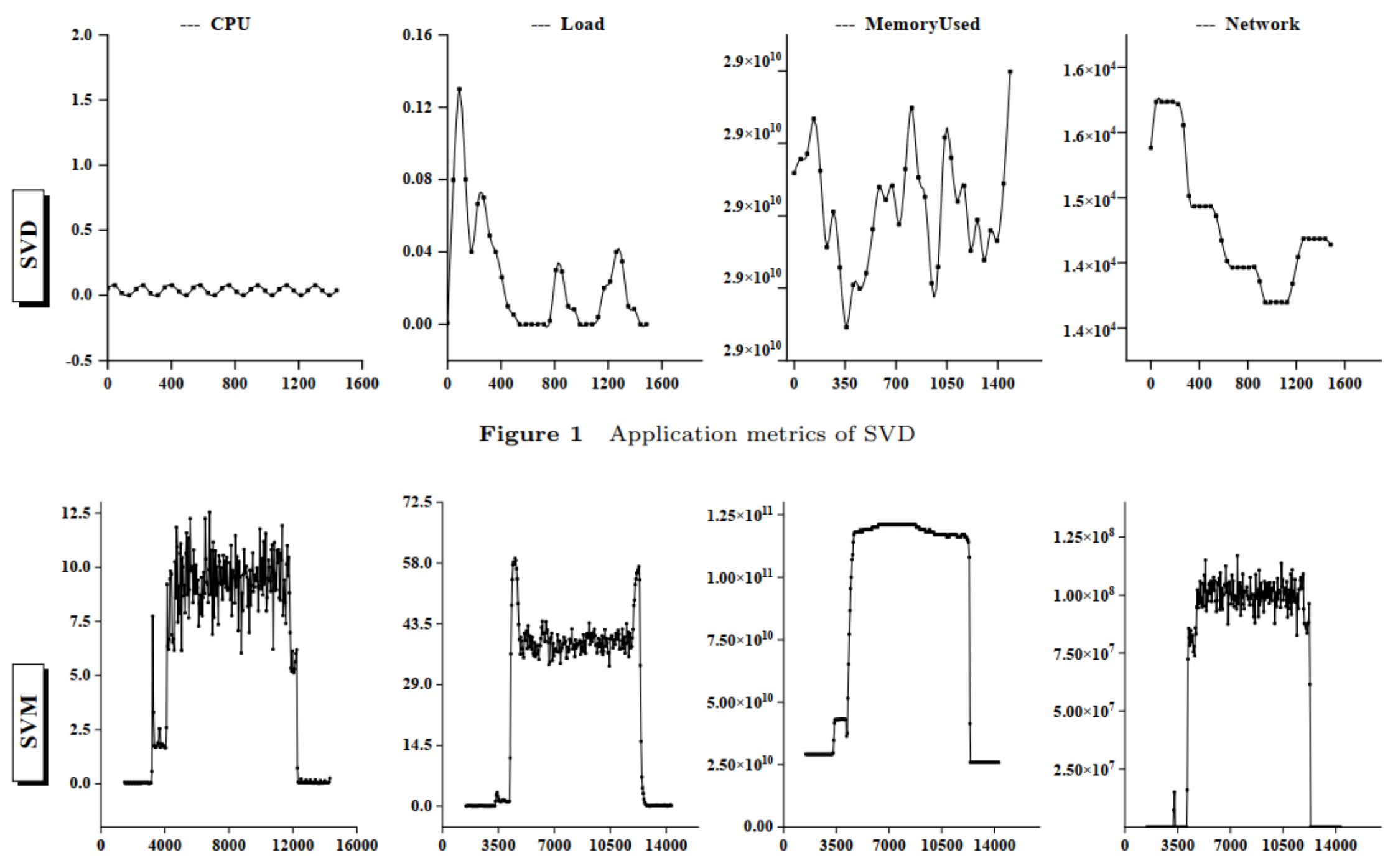


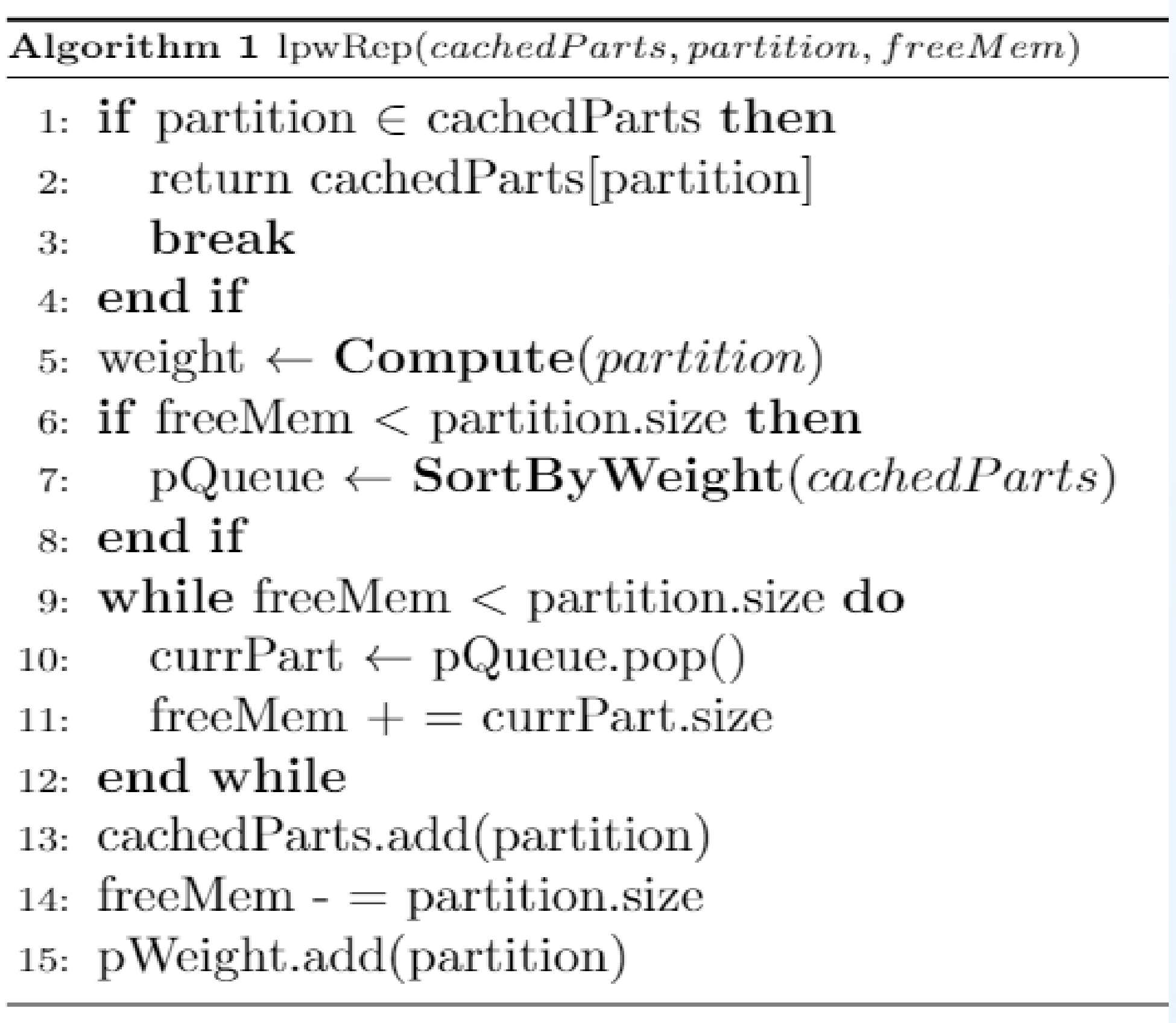
Figure 1 Application metrics of SVD

- **Diversity** of applications characteristics.
- **Variability** of memory resource requirement.
- **Uncertainty** of cache API usage.

LPW establishes a weight model based on factors to achieve effective use of cached data.

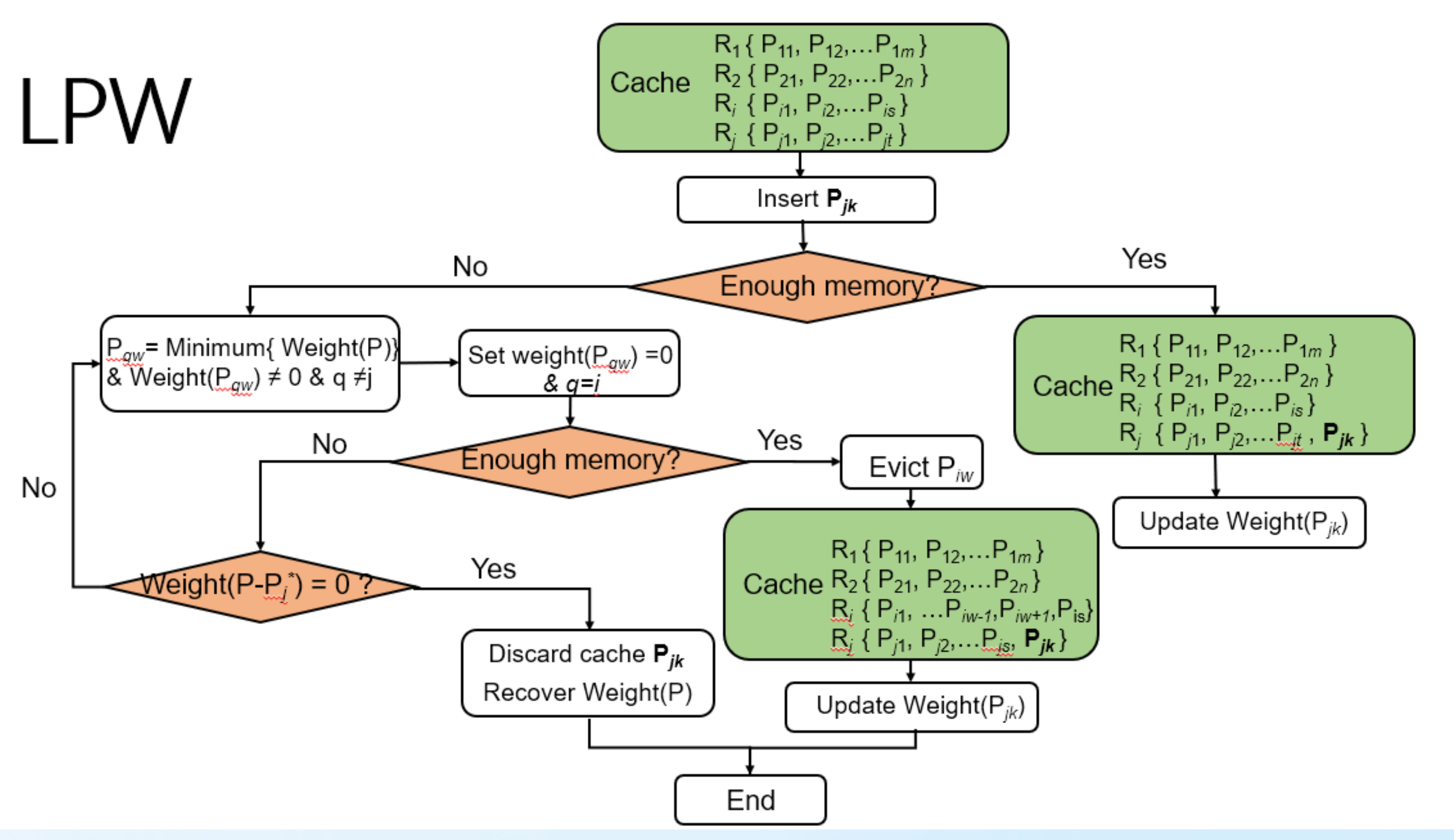
Algorithm

We take comprehensive consideration of different factors affecting performance, such as partition size, computation cost and reference count to find a most suitable partition to be replaced.


```
Algorithm 1 lpwRep(cachedParts, partition, freeMem)
if partition ∈ cachedParts then
    return cachedParts[partition]
    break
end if
weight ← Compute(partition)
if freeMem < partition.size then
    pQueue ← SortByWeight(cachedParts)
end if
while freeMem < partition.size do
    currPart ← pQueue.pop()
    freeMem + = currPart.size
end while
cachedParts.add(partition)
freeMem - = partition.size
pWeight.add(partition)
```

Methodology

A new block P_{jk} that belongs to RDD_j need to be cached. LPW dynamically check the weight of partitions and make reasonable replacement decision.


LPW

Cache $R_1\{P_{11}, P_{12},\dots P_{1m}\}$ $R_2\{P_{21}, P_{22},\dots P_{2n}\}$ $R_i\{P_{i1}, P_{i2},\dots P_{is}\}$ $R_j\{P_{j1}, P_{j2},\dots P_{jt}\}$

Insert $\mathbf{P_{jk}}$

Enough memory?

No

Yes

Cache $R_1\{P_{11}, P_{12},\dots P_{1m}\}$ $R_2\{P_{21}, P_{22},\dots P_{2n}\}$ $R_i\{P_{i1}, P_{i2},\dots P_{is}\}$ $R_j\{P_{j1}, P_{j2},\dots P_{jt}, \mathbf{P_{jk}}\}$

Update Weight(P_{jk})

P_{qw}= Minimum{ Weight(P)} & Weight(P_{qw}) ≠ 0 & q ≠j

Set weight(P_{qw}) =0 & q=i

Enough memory?

No

Yes

Evict P_{iw}

Cache $R_1\{P_{11}, P_{12},\dots P_{1m}\}$ $R_2\{P_{21}, P_{22},\dots P_{2n}\}$ $R_i\{P_{i1}, \dots P_{iw-1}, P_{iw+1}, P_{is}\}$ $R_j\{P_{j1}, P_{j2},\dots P_{js}, \mathbf{P_{jk}}\}$

Update Weight(P_{jk})

No

Weight(P-P_i^*) = 0 ?

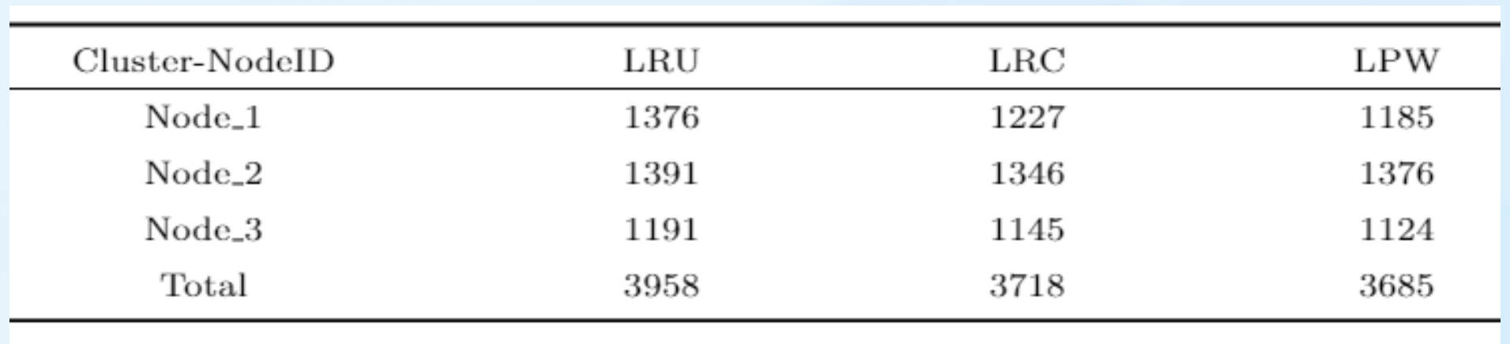
Yes

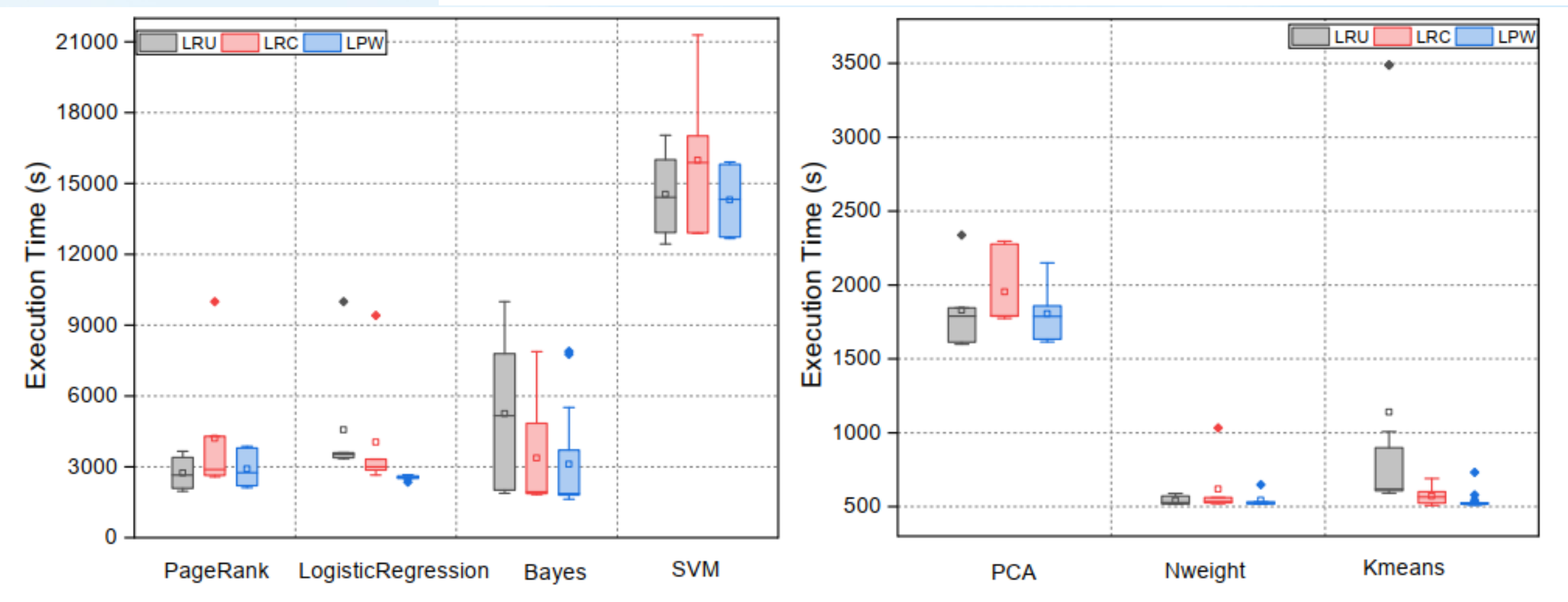
Discard cache $\mathbf{P_{jk}}$
Recover Weight(P)

End

Evaluation

We select workloads having *cache* API in HiBench[1]. LPW can speed up the execution of applications compared to LRU and LRC. Also, LPW could find hot data to keep in memory without causing frequent replacement.


Cluster-NodeID	LRU	LRC	LPW
Node_1	1376	1227	1185
Node_2	1391	1346	1376
Node_3	1191	1145	1124
Total	3958	3718	3685



Conclusion

- We deeply analyze multiple factors of cached partitions that affect application performance.
- We build a weight model to comprehensively evaluate the necessity based on various factors to achieve efficient use of cached data.
- We implement the cache replacement strategy LPW. Our comprehensive experiments show the effectiveness of LPW especially for iterative applications.

[1] "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis" (ICDEW 2010)