

Karma: Resource Allocation for Dynamic Needs

Mrinal Bhan¹, Priyanshu Soni², Mallikharjuna Rao K³

Abstract—We address the problem of fairly allocating resources in a system with dynamic user demands, specifically in big data processing systems. Current scheduling algorithms are effective for dealing with static workloads, but they struggle with the fluctuating resource demands inherent in big data analytics. To tackle this issue, we have introduced Karma, a new approach to resource allocation in dynamic big data environments. Karma uses a credit system, where users with resources not being used in the cluster can contribute to a pool, earning credits that can be used in the future. These credits can be used to acquire additional resources when a user's big data analysis job demands exceed their allocation. Karma prioritizes resource allocation based on user credits, ensuring fairness while optimizing cluster utilization. We demonstrate theoretically that Karma guarantees fairness, prevents manipulation, and achieves optimal resource efficiency even with dynamic user demands. Furthermore, real-world evaluations confirm that Karma minimizes user performance disparities while maximizing overall cluster efficiency for big data workloads.

Index Terms—Dynamic Resource Management, Big Data processing systems, Scheduling Algorithms, Optimal Cluster Utilization, Optimal Resource Efficiency

I. INTRODUCTION

Managing resources for big data is a challenging task that requires a unique approach due to the constantly changing needs of the users. Unlike traditional computing environments that have predictable demands, big data systems face constant fluctuations that can cause issues in resource allocation methods resulting in unfair distribution and inefficiencies. To address these complexities, Karma is introduced as a new approach specifically designed for big data processing.

Karma's credit system promotes a fairer distribution of resources than the existing methods. Despite the dangers that comes with handling big data, Karma ensures that all users receive resources fairly and equitably. By doing so, Karma encourages cooperation and collaboration within the big data processing environment. Moreover, users are not penalized for unexpected workloads, which fosters a sense of fairness and reliability.

Additionally, Karma has been proven to enhance the overall performance of the cluster. Conventional methods of resource allocation often fail to adapt to changing demands, which results in performance discrepancies between users. However, Karma dynamically adjusts the allocation of resources based on user requirements and credits, which substantially reduces

such performance disparities. This translates to a more efficient big data processing environment, where all users experience consistent and optimal performance levels.

Karma promotes a culture of resource sharing in the big data ecosystem by incentivizing users to contribute their unused resources to the pool. The credit system rewards them for their contribution, which not only optimizes overall cluster utilization but also fosters a collaborative environment where users benefit from each other's contributions. This approach is in contrast to traditional methods, which tend to encourage hoarding of resources, ultimately hindering the efficiency of big data processing as a whole.

Karma presents a groundbreaking approach to resource allocation in big data systems. It adapts to the ever-changing dynamics of big data workloads to ensure fairness, optimize performance, and foster resource sharing. By doing so, Karma establishes a more efficient and collaborative environment for processing big data, paving the way for a brighter future.

II. LITERATURE REVIEW


While extensive research exists on resource allocation and scheduling, comparing Karma to every individual work would be impractical. We focus on identifying the closest related works and highlighting Karma's unique contributions in the context of dynamic user demands.


1. Max-min Fairness Variants: Numerous studies explore variants of max-min fairness for cloud and cluster scheduling (1)(2)(3)(4). These approaches, similar to max-min fairness itself, assume static user demands. Karma tackles a distinct challenge: ensuring fairness and efficiency with dynamic demands. Generalizing Karma for multiple resources remains an open problem, but successfully addressing this limitation would necessitate a comparison with CARBYNE. However, CARBYNE also assumes non-strategic users and converges to max-min fairness in single-resource scenarios.


2. Fairness in Application Performance: Fairness in application-perceived performance is indirectly related to resource allocation fairness(5)(6). Other factors like hypervisors, storage systems, and resource preemption granularity can impact performance. Similar to existing mechanisms, Karma's properties are independent of these system-level factors. While our evaluation shows benefits at the application level, absolute numbers depend on the underlying system implementation.

3. Allocation of Time-Shared Resources: Generalized Processor Sharing (GPS) is an idealized algorithm for network link sharing, assuming infinitely divisible traffic. For specific conditions, GPS reduces to max-min fairness. However, GPS fairness guarantees only hold under specific assumptions about flow backlogs, which do not apply to dynamic user demands. Karma addresses this limitation by introducing new mechanisms for fairness under dynamic conditions.

4. Pricing and Credit-Based Allocation: Although they don't apply to Karma's use cases and don't concentrate

¹ Akshansh Singh is with the Data Science and Artificial Intelligence Department, IIIT-Naya Raipur, Chhattisgarh 493661, India (e-mail: akshansh21102@iiitnr.edu.in). 

² Mrinal Bhan is with the Data Science and Artificial Intelligence Department, IIIT-Naya Raipur, Chhattisgarh 493661, India (e-mail: mrinal21102@iiitnr.edu.in). 

³ Priyanshu Soni is with the Data Science and Artificial Intelligence Department, IIIT-Naya Raipur, Chhattisgarh 493661, India (e-mail: priyanshus21102@iiitnr.edu.in). 

³ Dr. Mallikharjuna Rao K is with the Data Science and Artificial Intelligence Department, IIIT-Naya Raipur, Chhattisgarh 493661, India (e-mail: mallikharjuna@iiitnr.edu.in).

Resource allocation is a fundamental problem

Allocating a resource (e.g. CPU, Memory) with a fixed capacity across multiple users

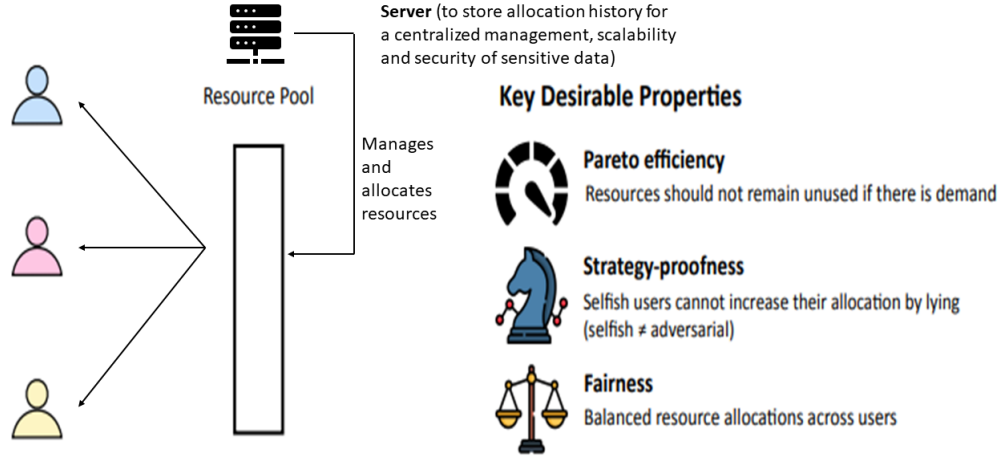


Fig. 1: Desirable Properties of resource allocation

on equitable resource allocation, pricing and bidding-based resource allocation techniques do exist(7). Though, unlike Karma, XChange puts forth a market-based strategy for multi-core architectures that puts immediate justice ahead of long-term justice. We are not aware of credit-based mechanisms addressing resource allocation with dynamic user demands, even though credits are used in game theory contexts.

Karma's Distinction: Karma stands out by specifically addressing the challenge of fair and efficient resource allocation in big data systems with dynamic user demands. It is different from existing approaches that assume static demands and introduces a novel credit-based system to achieve fairness and optimal resource utilization under dynamic conditions.

III. METHODOLOGY

This section delves into the design and execution of Karma, a resource allocation mechanism specifically crafted to address the complexities of resource allocation in big data processing environments characterized by dynamic user demands. Karma leverages a credit-based system to incentivize resource sharing and achieve a balance between fairness and efficiency. The following subsections will detail the core components of Karma, including the resource allocation algorithm that orchestrates resource and credit exchange among users, the theoretical underpinnings that guarantee fairness and efficiency under dynamic demands, and the evaluation methodology employed to assess Karma's performance within a big data processing cluster.

A. Karma

Karma tackles resource allocation for big data systems with dynamic user demands. It uses a credit system to ensure fairness and efficiency. Users with low demands "donate" excess resources (credits) to a pool. Users with high demands can "borrow" resources (spend credits) from this pool. Karma prioritizes allocation to balance credit distribution (donors) and maintain fair resource allocation (borrowers). We theoretically prove Karma achieves Pareto efficiency, strategy-proofness, and fairness for dynamic demands.

B. Preliminaries

We consider n users sharing a single resource divided into quanta (time slices). Users have a fair share (f) of resource units (CPU, memory, GPUs, etc.); per quantum. Karma allocates resources at the beginning of each quanta. User demands are arbitrary per quanta and don't carry over. Users are strategic (maximize allocations) but not adversarial (don't lie).

C. Karma Design

Karma guarantees each user an portion of their fair share ($\alpha * f$) per quanta (guaranteed share). It maintains a pool of resource slices (karmaPool) with two types:

- Shared slices: not guaranteed to any user (total: $n * (1 - \alpha) * f$)
- Donated slices: contributed by users with demands below their guaranteed share.

In a quanta, users with demand below their guaranteed share donate the difference. Users with demand exceeding their guaranteed share borrow from shared or donated slices.

D. Karma Credits

Karma allocates resources based on both demand and past allocations (credits). Users earn credits in three ways:

- 1) Initial credits upon joining the system.
- 2) $(1 - \alpha) \cdot f$ Free credits every quanta for contributing $(1 - \alpha)$ fraction of its fair share to shared slices
- 3) One credit per borrowed donated slice (per quanta).

Unlike earning credits, there is only one way for any user to lose credits: for every slice borrowed from the karmaPool (donated or shared), the user loses one credit

E. Prioritized Resource Allocation

We now present the resource allocation algorithm of Karma, which coordinates credits and resources among users (Algorithm 1). Assuming that user demands exceed their guaranteed share and totaling the difference between their demand and $\alpha \cdot f$ in a given quanta, we define "borrower demand." All slices

are assigned to borrowers, and credits are updated as previously mentioned, when borrower demand precisely matches the available pool (karmaPool). Coordination of resources and credits when supply and borrower demand are different presents the main algorithmic problem.

Supply Exceeds Borrower Demand: Here, karmaPool has enough slices to pay off every borrower. Choosing slices from donors with the fewest credits comes first in Karma's priority allocation of donated slices. Through this method, users with smaller credit balances—which may have been under-allocated in the past—can earn more credits, so encouraging a more equitable credit distribution among all users. Credit balances naturally show a user's allocation history; those with consistently lower allocations will have higher credit balances than those with surplus allocations. Karma works to get a more fair overall allocation for all users by balancing credit distribution. When every donated slice is used up, Karma distributes the remaining borrower demand among shared slices.

Supply Less Than Borrower Demand: When there aren't enough karmaPools to satisfy borrowers, Karma uses a method to give users with larger credit balances slices. Using their higher credit balances, users who have previously received less resources are more favoured, which over time promotes an equal resource distribution.

Algorithm 1 Karma Dynamic Resource Allocation

```

1:  $demand[u]$ : user  $u$ 's demand
2:  $credits[u]$ : user  $u$ 's current credits
3:  $alloc[u]$ : user  $u$ 's allocation provided
4:  $f$ : fair share
5:  $\alpha$ : guaranteed share
6: for ( do
    Each quantum)  $slices\_shared \leftarrow n \cdot (1 - \alpha) \cdot f$  for
    ( do every user  $u$ )
7:    $credits[u]$  incremented by  $(1 - \alpha) \cdot f$ 
10:   $slices\_donated[u] = \max(0, \alpha \cdot f - demand[u])$ 
11:   $alloc[u] = \min(demand[u], \alpha \cdot f)$ 
12: end for
13:  $donors \leftarrow$  all users  $u$  with  $slices\_donated[u] > 0$ 
     $oborrowers \leftarrow$  all users  $u$  with
         $alloc[u] < demand[u] \& credits[u] > 0$ 
14: while ( do
    borrowers  $\neq \emptyset$  and  $(\sum_u slices\_donated[u] > 0$  or  $shared\_slices > 0)$ )  $b^* \leftarrow$  borrower with
    max credits if ( then  $donors \neq \emptyset$ )
15:    $d^* \leftarrow$  donor with min credits
16:   Increment  $credits[d^*]$  by 1
17:   Decrement  $slices\_donated[d^*]$  by 1
18:   Update donors (line 13)
19: else
20:   Decrement  $slices\_shared$  by 1
21:   Increment  $alloc[b^*]$  by 1
22:   Decrement  $credits[b^*]$  by 1
23: end if
24:   Update borrowers (line 14)
25: end while
26: end for=0

```

IV. EXPERIMENT & RESULTS

A. Experimental Setup

We tried to compare karma resource allocation algorithm with the existing algorithm used in practice i.e. max-min fairness.

The simulation environment consists of the following:

- Number of Users: $N = 3$
- Simulation Duration: $T = 10$ seconds
- Resource Pool: R_T (fixed amount of resource units) = 6 slices

Each user's demand fluctuates dynamically:

- User 1: Low demand generally with occasional spikes in demand
- User 2: High demand throughout with a few low demands
- User 3: Constant moderate demand throughout the simulation.

Table 1 provides the demand of each user overtime.

TABLE I: Demands of Each User Over Time

User	1s	2s	3s	4s	5s	6s	7s	8s	9s	10s
user1	1	3	2	1	1	1	4	2	1	5
user2	4	5	1	5	5	2	1	5	3	1
user3	3	1	4	4	2	4	5	1	2	1

B. Results

Now let us see how the existing method of max-min fairness would allocate resources in this:

TABLE II: Max-Min allocation

User	1s	2s	3s	4s	5s	6s	7s	8s	9s	10s
user1	1	2	2	1	1	1	2	2	1	4
user2	3	3	1	3	3	2	1	3	3	1
user3	2	1	3	2	2	3	3	1	2	1

Comparing this with karma algorithm for resource allocation:

TABLE III: Karma allocation

User	1s	2s	3s	4s	5s	6s	7s	8s	9s	10s
user1	1	3	2	1	1	1	3	2	1	4
user2	2	2	1	3	3	2	1	3	3	1
user3	3	1	3	2	2	3	2	1	2	1

Let us plot some graphs to compare which method works better.

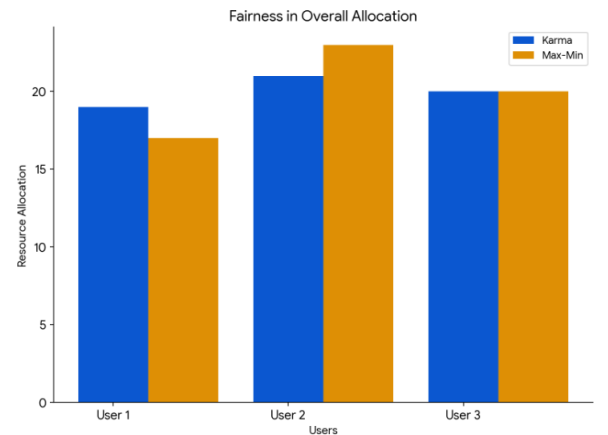


Fig. 2: Overall Resource Allocation

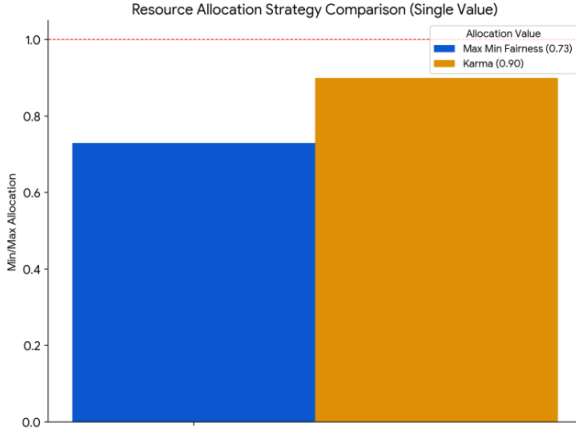


Fig. 3: (Min/Max) Allocation Value

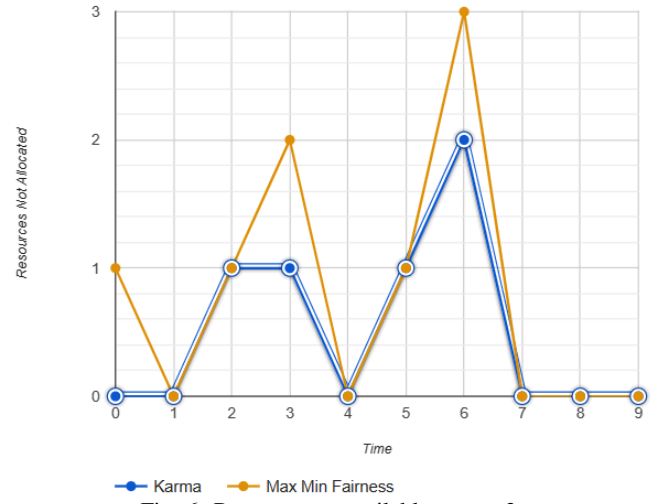


Fig. 6: Resources unavailable to user3

Also let us see the disparity in the resources allocated to each user.

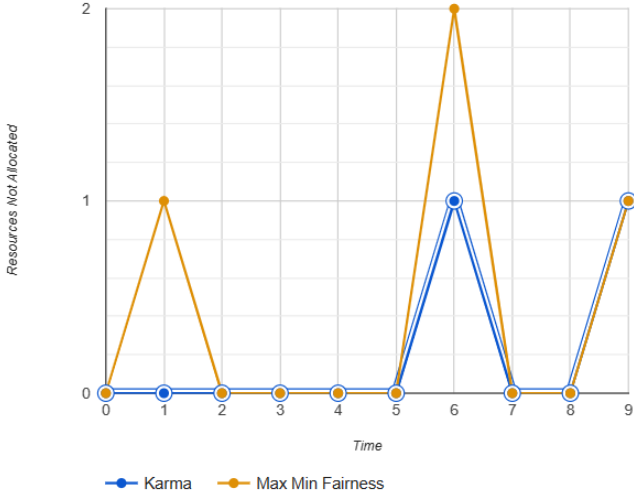


Fig. 4: Resources unavailable to user1

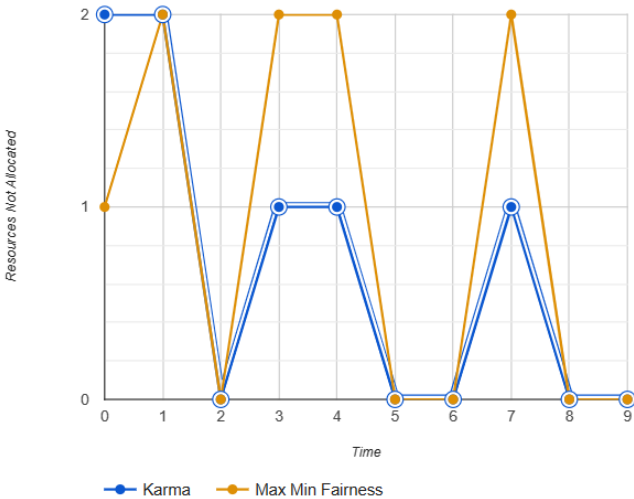


Fig. 5: Resources unavailable to user2

C. Discussion

- **Fairness Through User Contribution:** Max-min fairness focuses on immediate fairness and doesn't consider the overall fairness of resources allocated. This is done in a more efficient way in karma since it considers credits and rewards users with a higher credit value ensuring an equal distribution in the resources allocated overtime.
- **Addressing Demand Fluctuations:** Max-min fairness does not perform as well in scenarios where users have fluctuating demands. In such a scenario if everyone has a high demand the algorithm ends up taking away resources from those with a low demand. However, this situation can be avoided in karma since it prioritizes users with a high credit value instead of a higher demand making it a more fair distribution
- **Trade-offs and Considerations:** Even though karma is more efficient in resource allocation and making the distribution fair, it is more complex compared to max-min fairness since it involves an additional aspect of managing credits and how users gain lose and spend them.

V. CONCLUSION AND FUTURE WORKS

This work addressed the limitations of the classic max-min fairness algorithm in dynamic user demand scenarios. Max-min fairness struggles to maintain desirable properties like Pareto efficiency, strategy-proofness, and fairness when user demands fluctuate. We proposed Karma, a novel resource allocation mechanism specifically designed for dynamic user demands. We provided theoretical proofs demonstrating that Karma guarantees Pareto efficiency, strategy-proofness, and fairness in these dynamic environments.

Furthermore, experimental evaluation within a multi-tenant elastic memory system confirmed that Karma's theoretical strengths translate to practical benefits. Compared to max-min fairness, Karma significantly reduces application-level performance disparity (up to 2.4x) while maintaining high resource utilization and overall system performance.

Karma opens doors for exciting future research directions. These include extending Karma's theoretical analysis for a broader range of α values, generalizing Karma to handle multiple resource types, and adapting it to accommodate all-or-nothing or gang-scheduling constraints common in GPU

resource allocation. By addressing the challenges of dynamic user demands, Karma paves the way for more efficient and equitable resource allocation in big data processing systems.

VI. ACKNOWLEDGMENT

We would like to express our gratitude to Dr. Mallikharjuna Rao K for all of his help and assistance with this endeavor. We also value the academics and staff at our university for helping to provide facilities and resources. We are appreciative of the chance to work on this project since it has been a worthwhile educational experience.

REFERENCES

- [1] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. Balancing Efficiency and Fairness in Heterogeneous GPU Clusters for Deep Learning. In EuroSys, 2020.
- [2] Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A GPU Cluster Manager for Distributed Deep Learning. In NSDI, 2019.
- [3] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. Themis: Fair and Efficient GPU Cluster Scheduling. In NSDI, 2020.
- [4] Deepak Narayanan, Fiodar Kazhamiaka, Firas Abuzaid, Peter Kraft, Akshay Agrawal, Srikanth Kandula, Stephen Boyd, and Matei Zaharia. Solving Large-Scale Granular Resource Allocation Problems Efficiently with POP. In SOSR, 2021.
- [5] Tao Luo, Mingen Pan, Pierre Tholoniati, Asaf Cidon, Roxana Geambasu, and Mathias Lécuyer. Privacy Budget Scheduling. In OSDI, 2021.
- [6] Chengzi Lu, Kejiang Ye, Guoyao Xu, Cheng-Zhong Xu, and Tongxin Bai. Imbalance in the Cloud: An Analysis on Alibaba Cluster Trace. In Big Data, 2017.
- [7] Rich Wolski, John Brevik, Ryan Chard, and Kyle Chard. Probabilistic Guarantees of Execution Duration for Amazon Spot Instances. In SC, 2017.