Revisiting Pebble Games for Modeling and Efficient Use of Disaggregated Memory Systems

Anusha Devulapally
The Pennsylvania State University
USA
akd5994@psu.edu

Mahantesh Halappanavar
Pacific Northwest National Laboratory
USA
hala@pnnl.gov

Bruno Jacob
Pacific Northwest National Laboratory
USA
bruno.jacob@pnnl.gov

Vijaykrishnan Narayanan
The Pennsylvania State University
USA
vijaykrishnan.narayanan@psu.edu

Andres Marquez
Pacific Northwest National Laboratory
USA
Andres.Marquez@pnnl.gov

ABSTRACT

Introduced in 1981, the Red-Blue game of Hong and Kung uses red (representing limited but fast memory) and blue (representing unlimited but slow memory) pebbles to compute the I/O complexity of an application represented as a directed acyclic graph. Several variants of this game have been designed to address different aspects of I/O performance and system configurations. In contrast, disaggregated memory systems have recently emerged as a means to address poor utilization of memory resources, especially in the context of cloud computing. In this position paper, we posit that pebble games are an effective tool to model performance, as well as design, effective disaggregated memory systems. We introduce a novel variant of the game, **Dy**namic red-orange-blue **Pe**bble **G**ame with multiple processors (DyPeG), to reflect dynamic provisioning of memory resources that is appropriate for multi-tenant cloud computing environments. Using representative real world applications, we provide empirical evaluation to demonstrate the effectiveness of the proposed approach to study disaggregated memory systems.

KEYWORDS

Pebble Games, CXL, Red-Blue Pebble Game, Multi-tenant Cloud Environments, Disaggregated Memory System, Fabric Attached Memory, I/O complexity, Memory Scheduling, Computational Graphs, Multiple Processors, Greedy Algorithm, Dynamic Modeling

1 INTRODUCTION

Given a directed acyclic graph (DAG), $G = (V, E)$ with vertex set V and edge set E, a *red-blue pebble game* is a single player game that proceeds by placing colored pebbles on vertices using a set of rules. The color of a pebble represents the type of memory used. The red-blue pebble game introduced in the seminal work of Hong and Kung [11] has the following rules: (i) Input: Blue pebbles are placed on vertices with no predecessors (incoming edges); (ii) Movement: Red pebbles can be placed on any vertex with a blue pebble; (iii) Compute: A vertex can be computed if all its predecessors have a red pebble; and (iv) Output: Blue pebbles are placed on all terminal nodes (no outgoing edges), after which the game terminates. This game was introduced to model the I/O complexity of algorithms on a two-level memory hierarchy and is known to be NP-hard in general [5]. However, greedy algorithms with approximation bounds perform well in practice [15]. We detail the game and its variants in §4.

Memory disaggregation is an approach to pool memory resources across multiple computing nodes (processors) to enable applications to access shared memory dynamically. Disaggregation has emerged as a foundational technique to address underutilization of memory resources especially on cloud computing platforms, as well as to scale memory-intensive applications by providing access to large shared pools of memory. Given the scale of operations, efficient use of cloud computing resources can lead to significant reduction in costs [9]. Memory pooling can improve memory utilization, reduce fragmentation, and thus lower overall infrastructure costs. Advances in high-bandwidth interconnects and fabric-attached memory architectures have made disaggregated memory systems attractive. Open industry standards such as Compute Express Link (CXL) [4] are now driving wider adoption [10]. However, tools for modeling, analyzing, and designing efficient disaggregated systems still remain underdeveloped. This poses a significant challenge for adoption, which we aim to address in this work.

We posit that pebble games are an effective tool for modeling disaggregated memory systems and for designing efficient systems tailored to specific application use cases and system parameters. Furthermore, algorithms for pebble games can be exploited to develop dynamic memory allocation strategies that can also meet service-level agreements while maximizing for performance and data movement costs. Towards this end, we introduce a novel variant of the pebble games, *dynamic red-orange-blue pebble game with multiple processors* (DyPeG), to address the practical needs of cloud computing platforms to support efficient use of memory pools in a multi-tenant setting with dynamic allocation of resources and changes in performance due to congestion, while subject to service level agreements (detailed in §3). We design a greedy algorithm to solve DyPeG and perform empirical evaluation of the algorithm using a representative workload. We present the details in §5.

The main contributions of this work are:

- Introduce the dynamic red-orange-blue pebble game with multiple processors (DyPeG).
- Develop greedy algorithm to efficiently solve multiprocessor and multi-level pebbles games, including DyPeG.
- Perform preliminary empirical evaluation of DyPeG using representative computational DAGs for Fast Fourier Transform (FFT) and residual neural network (resnet).

To the best of our knowledge, this is the first work to introduce pebbles games in a dynamic setting, as well as targeting disaggregated memory systems.

2 PRELIMINARIES

Computational problems can be modeled as directed acyclic graphs (DAGs), where the vertices represent individual computational tasks and the edges encode the dependencies between the tasks. In memory-constrained settings, the efficiency of a computation is heavily influenced by its I/O complexity, which quantifies the number of data transfers occurring across various levels of the memory hierarchy (e.g., cache, local memory, remote memory, and secondary storage). In classical computational models, I/O complexity provides lower bounds on the number of memory transfers necessary to execute an algorithm given a specified amount of fast memory. For instance, performing an n-point Fast Fourier Transform (FFT) using $O(S)$ memory incurs a minimum I/O cost of $\Omega\left(n \log n / \log S\right)$, while matrix multiplication for an $n \times n$ matrix has an I/O lower bound of $\Omega\left(n^3/S\right)$. These theoretical limits underscore the critical role played by memory capacity, latency, and bandwidth, particularly in disaggregated-memory systems where these parameters are substantially influenced.

We briefly introduced the classical red-blue pebble game in §1, which can be extended to consist of multiple memory types (multi-level), as well as multiple processors. We now introduce a multi-processor red-orange-blue pebble game. In order to enable multi-processing, we consider different *shades* of red pebbles, R^i, $1 < i < P$, representing local memory on P processors. The total number of red pebbles is the sum of local memories on each processor, $\sum_{i=1}^{P} R^i$. For the sake of this discussion, we consider a distributed-memory system with access to a shared (fabric attached) memory pool. The pooled memory is represented with orange pebbles, O. We assume that the cost for accessing data from pooled memory is much lower than the cost of accessing remote memory, where remote memory can be shared via a message passing framework. Figure 1 provides a simplified illustration of such a system. We propose the following rules for a multiprocessor red-orange-blue pebble (MROBP) game:

- R1 (Input) Assign blue pebbles to all source or input nodes (vertices with no predecessors).
- R2 (Remote Get): An orange pebble or a red pebble of a different shade (R^q, $q \neq p$, when processor p works on the given vertex) can be placed on any vertex that already has a blue pebble.
- R3 (Data Movement): A red pebble of any shade can be placed on any vertex that already has a blue, orange, or a red pebble of a different shade.
- R4 (Compute) A vertex can be computed if all its immediate predecessors have red pebbles. Place a red pebble of any shade on this vertex. For all predecessors with a different shade of red will incur a remote data movement cost.
- R5 (Remote Put) An orange pebble or a red pebble of a different shade (R^q, $q \neq p$) can be placed on any vertex that has a red pebble R^p.
- R6 (Output) A blue pebble can be placed on any vertex that has a red pebble.
- R7 (Termination) The game terminates when no further moves can be made, or when all terminal nodes (vertices with no successors) are computed and placed in memory.

We note that a multi-level, multi-processor game is still a static game, in that the system remains stable. We extend this game with a dynamic access to resources, where the volume and performance of memory system can change over the execution of a given DAG.

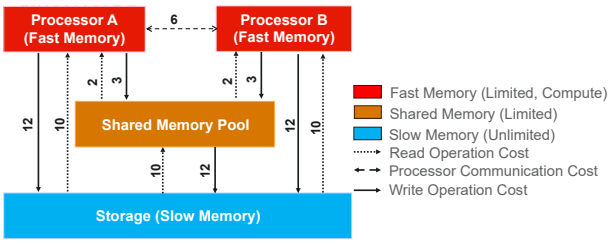

Figure 1: Multi-Level, multi-processor memory hierarchy and data movement costs in a disaggregated system. We assume similar costs for moving from Blue to Red and Blue to Orange, but much smaller costs for moving from Orange to Red. We also assume different costs for Read versus Write tasks. We also assume that the message passing between the processors is more than moving it to Orange. These cost values are only representative numbers for empirical evaluation.

3 DYNAMIC MULTILEVEL PEBBLE GAME

We introduce the dynamic multi-level pebble game to specifically address multi-tenant cloud computing systems with disaggregated memory where memory resources are dynamically provisioned over the period of execution of one or more applications.

The dynamic extension builds on the MROBP discussed in the §2 by introducing the time-varying constraints on the availability of red pebbles, mimicking how memory quotas are adjusted at runtime in shared cloud infrastructures. The red pebble budget, representing fast memory or compute-local capacity, can change during execution due to external system factors such as tenant interference or adaptive provisioning. To ensure fair resource guarantees, a Service Level Agreement (SLA) is enforced by maintaining a lower bound on the red pebble budget. This SLA threshold is determined through an offline iterative approach, where we gradually increase the red pebble count and simulate greedy execution until the DAG completes without stalling. The resulting minimum working red pebble count provides a baseline for dynamic adjustment: any temporary budget changes must respect this SLA to guarantee progress. This formulation enables analysis of pebbling strategies under realistic constraints where memory cannot be statically provisioned. An additional constraint that can be enforced to reflect conditions such as congestion would be to change the costs of memory accesses dynamically, which we will perform in the future.

The rules for dynamic extension builds on the MROBP in DyPeG can be formalized as follows:

- D1 (Input) Assign blue pebbles to all source or input nodes (vertices with no predecessors).
- D2 (Dynamic Budget Constraint) A red pebble can only be placed if the total number of red pebbles does not exceed the current budget. This budget must remain at or above a specified SLA-defined minimum. If the red-pebble limit is reached, an existing red pebble must be removed before placing a new one.

D3 (Remote Get) An orange pebble or a red pebble of a different shade (R^q, $q \neq p$, when processor p computes the vertex) can be placed on any vertex that already has a blue pebble.
D4 (Data Movement) A red pebble of any shade may be placed on a vertex that currently has a blue pebble, an orange pebble, or a red pebble of a different shade. If the vertex has no existing pebble, the placement counts as a fresh memory fetch.
D5 (Compute) A vertex can be computed if all its immediate predecessors have red pebbles. Place a red pebble R^p on this vertex. For all predecessors holding red pebbles of a different shade (R^q, $q \neq p$), the step incurs a remote data movement cost.
D6 (Remote Put) An orange pebble or a red pebble of a different shade (R^q, $q \neq p$) can be placed on any vertex that currently has a red pebble R^p. This models explicitly exporting data for reuse by other processors or into slower memory.
D7 (Eviction) A red pebble may be removed (converted to a blue pebble) to stay within budget. This is allowed if the node:
- is a sink node,
- has all its successors already computed, or
- has successors that all hold blue or orange pebbles.

D8 (Output) A blue pebble may be placed on any vertex that has a red pebble, marking its result as persisted in memory.
D9 (Termination) The game terminates when no further moves can be made, or when all sink nodes are computed and finalized in memory.

We evaluate DyPeG using two representative applications and present the results in §5.

4 RELATED WORK

The black pebble game was introduced over five decades ago by Sethi to model the space complexity of registers to execute computer programs [19]. It modeled memory (register) usage on a DAG generated from straight-line programs, where memory slots (registers) were modeled as pebbles. The goal was to execute the program using the smallest amount of memory (number of registers). In this game, pebbles can be placed on source nodes anytime, or on a node once all its predecessors are pebbled. Pebble can be removed freely. Sethi's register-allocation problem is equivalent with the pebbles corresponding to registers holding computed values without recomputation [19]. The progressive variant forbids the recomputation, i.e., each node pebbled at most once, making the decision problem NP-Hard, whereas allowing recomputation is PSPACE-complete [7]. A further black-white pebble game adds white pebbles to model non-deterministic guesses [1, 13]. Placing a white pebble on a node corresponds to guessing its value, which must later be verified by actually computing the node. While this approach can save at most one pebble, it may square the number of moves required.

Petri net, a mathematical representation to model and understand concurrent interactions among different components of a system, is a closely related topic [16]. Since their introduction by Carl Adam Petri in their 1962 doctoral dissertation [20], Petri nets have been widely used in system and process modeling, software design, optimization, and memory systems [21]. Dijkstra introduced a nondeterministic pebble game to reason about program control and termination. Reisig later formalized Dijkstra's approach using Petri nets and proposed distributed, online and reversed variants, highlighting the algorithm's local and reversible properties [17]. Although not modeling memory or I/O directly, this work influenced formal reasoning about concurrency in pebble-like systems.

In their seminal work, Hong and Kung introduced the red-blue pebble game to model the I/O complexity of the algorithms [11]. This game modeled the trade-off between computation and memory usage for two-level memory hierarchy. Demaine and Liu further studied the complexity of computing optimal cache and memory transfer trade-offs [5], and Gleinig and Hoefler introduced red-blue-white pebble game for trees and larger DAGs refining the I/O lower bounds for large-scale computation [8]. In extending the classic two-level red-blue pebble game, Savage generalized the model to an arbitrary k-level hierarchy and derived matching I/O lower bounds and optimal scheduling principles across all levels [18]. Carpenter et al. introduced a one-shot variant of the pebble game, where each vertex may be pebbled only once [3].

Recent studies further extended red-blue pebble to multi-processor. Elango et al. developed a parallel extension of the classic red-blue pebble game, modeling multi-node, multi-core machines with hierarchical caches and interconnects to derive lower bounds on data-movement complexity of arbitrary CDAGs executed in parallel [6]. Liu et al. surveyed both black and red-blue pebble games in sequential and parallel settings to explore applications from register allocation to I/O modeling [14]. Kwasniewski et al. revisited red-blue pebbling to derive tight sequential and parallel I/O bounds for classic matrix-matrix multiplication, demonstrating how optimized pebbling strategies can significantly reduce communication costs in HPC environments[12]. Finally, Böhnlein et al. [2] propose a multiprocessor generalization of the red-blue pebble game that captures the trade-offs between load-balancing, communication and memory by proposing a greedy heuristic approach.

It is evident that pebble games have a long history of theoretical development. However, most work remains analytical or simulation-based and only a few studies incorporate empirical evaluation, either through simulated pebbling or benchmarking on real DAGs [2, 3, 12]. Empirical work is sparse, and there remains a significant gap in validating pebble games for modern workloads under dynamic system constraints. From an applications perspective, early pebble game research focused on structured computations like FFT and matrix multiplication [11, 18]. Later studies have applied pebble models to a wide range of domains, which include sparse matrix computations [22], deep neural networks [23], TSP graphs [3], and large random or tree-structured DAGs [8]. This trend reflects a shift towards modeling realistic workloads and capturing the computation–communication trade-offs in modern dataflow applications.

The existing variants of the pebble game assume that memory resources and performance (bandwidth and latency) remain static for the entire period of execution. However, this is assumption is not reflective of real-world, especially in the context of multi-tenant cloud computing systems. We therefore introduce a new variant, dynamic multi-level pebble game, to model modern systems and evaluate its performance for a representative AI workload. Furthermore, in addition to establishing theoretical bounds on I/O complexity, there is a need for a sound algorithmic approaches that can provide runtime guidance for dynamic memory allocation strategies for multi-tenant systems that host a heterogeneous mix of applications and are subject to service-level agreements.

5 WORKLOAD ANALYSIS AND RESULTS

We present detailed empirical evaluation using two representative algorithms, Fast Fourier transform (FFT) and Residual Network (resnet-coarse), under both the Red-Blue (RB) and Red-Orange-Blue (ROB) pebble game models. Using greedy scheduling and empirically determined working red pebble counts, starting with the theoretical lower bound of $\Delta + 1$, where Δ is the maximum in-degree. We quantify the total number of pebble moves and their associated costs across varying levels of processor parallelism and disaggregated-memory (orange pebbles) usage. We perform two types of evaluations: (*i*) keeping total memory constant, and (*ii*) keeping per-processor memory constant. We can expect to see additional costs from memory movement due to inter-process communication when the total memory is held constant. With increased total memory, we expect the per-node constant memory case to favor the increase in the number of processors. In this section, we present the results for FFT and then for resnet.

5.1 Fast Fourier Transform (FFT)

The FFT workload reveals important distinctions in how cost scales under Red-Blue (RB) and Red-Orange-Blue (ROB) pebble games.

5.1.1 ***Analysis 1: Keeping the total memory constant.*** In the RB model (Figure 2), total cost increases steeply with FFT problem size, and higher processor counts do not consistently reduce cost. Notably, the p=16 (16 processors) configuration performs worse than both p=1 and p=32 for larger problem sizes, suggesting suboptimal reuse and parallel scheduling under certain configurations. In contrast, the ROB model (Figure 3) demonstrates smoother cost trends, with consistent reductions as both problem size and processor count grow. By introducing pooled memory in the form of orange pebbles (8× red), ROB enables better intermediate value retention, significantly lowering total cost for large FFTs like 1024 and 2048. Across single- to 8-processor configurations, ROB reduces total cost by about 50–55 %, demonstrating a clear efficiency advantage over the classic RB schedule for all problem sizes (128–2048). The benefit narrows as we scale up—dropping to 37 % at 16 processors and 9 % at 32—because the gain from orange buffering is largely swallowed by the overhead of coordinating so many processors, so ROB delivers only marginal extra savings at that point. Table1 lists the total number of moves required for FFT DAGs across problem sizes and processor configurations.

5.1.2 ***Analysis 2: Keeping per-processor memory constant.*** As seen in the RB and ROB model (Figures 4 and 5), the total cost for all the cases is less compared to Analysis 1 (constant memory). This is due to the increase in total memory with increase in the number of processors. Similar to Analysis 1 total cost for RB increases steeply with FFT problem size, and higher processor counts consistently reduce cost. In contrast, the ROB model (Figure 5) demonstrates substantial efficiency gains for all the problem sizes over RB. Table 2 lists the total number of moves required for FFT DAGs across problem sizes and processor configurations.

5.2 Resnet-Coarse

The resnet-Coarse workload highlights the cost behavior of deep learning DAGs at a layer-wise granularity of the resnet models under RB and ROB pebble models. For this workload, we only perform

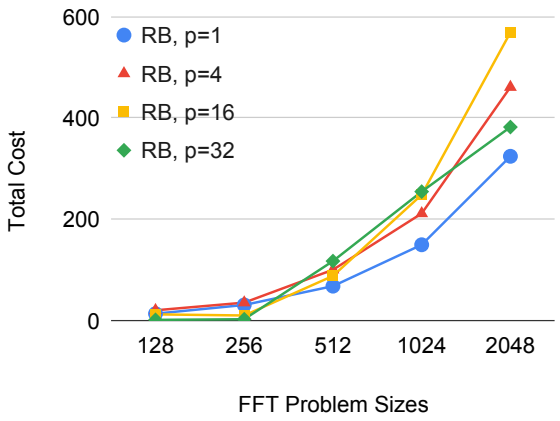

Figure 2: *RB Total Cost - FFT (Analysis 1):* Total cost (scaled by a factor of 1000) under the Red-Blue (RB) pebble game for FFT DAGs across varying processor counts keeping the total memory constant.

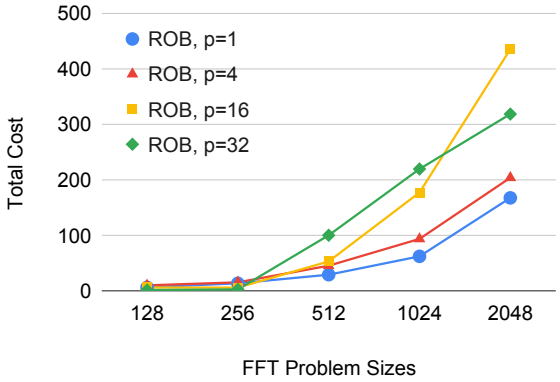

Figure 3: *ROB Total Cost - FFT (Analysis 1):* Total cost (scaled by a factor of 1000) for FFT DAGs under Red-Orange-Blue (ROB) pebble games across varying processor counts keeping the total memory constant.

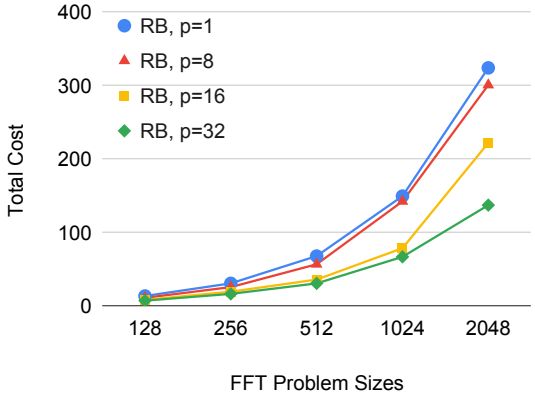

Figure 4: *RB Total Cost - FFT (Analysis 2):* Total cost (scaled by a factor of 1000) under the Red-Blue (RB) pebble game for FFT DAGs across varying processor counts keeping per-processor memory constant.

Problem Size	nodes	edges	in_degree	upper bound	min_red	Single Processor		2 Processors		4 Processors		8 Proc		16 Processors		32 Processors	
						RB	ROB	RB	ROB	RB	ROB	RB	ROB	RB	ROB	RB	ROB
128	1024	1792	2	8	146	1966	1983	2354	2416	2723	2775	1496	1496	1549	1549	128	128
256	2304	4096	2	8	275	4477	4479	2312	2312	4744	4744	1064	1064	1224	1224	256	256
512	5120	9216	2	8	634	9963	9983	11745	11745	13564	13564	14537	14537	11764	11764	15479	15479
1024	11264	20480	2	8	1281	22015	22015	24825	24825	28935	28935	31309	31309	33013	33013	33675	33675
2048	24576	45056	2	8	2179	47999	47999	55224	56019	63332	64008	71239	71262	75279	75279	50259	50259

Table 1: *FFT Greedy Total Moves for Analysis 1:* Total number of moves computed using the greedy algorithm for different FFT DAGs under Red-Blue (RB) and Red-Orange-Blue (ROB) pebble game models, with varying processor counts. ROB uses 8× red pebbles for orange memory.

Problem Size	nodes	edges	in_degree	upper bound	min_red	Single Processor		2 Processors		4 Processors		8 Proc		16 Processors		32 Processors	
						RB	ROB	RB	ROB	RB	ROB	RB	ROB	RB	ROB	RB	ROB
128	1024	1792	2	8	146	1966	1983	1961	2068	1963	2082	1959	2033	1747	1811	1663	1663
256	2304	4096	2	8	275	4477	4479	4602	4604	4540	4593	4417	4532	4030	4030	3797	3797
512	5120	9216	2	8	634	9963	9983	10084	10124	10307	10346	10015	10015	8393	8393	7983	7983
1024	11264	20480	2	8	1281	22015	22015	22270	22270	22624	22624	23482	23482	18533	18533	17605	17605
2048	24576	45056	2	8	2179	47999	47999	48196	48240	48972	49365	49942	51227	45045	46053	37879	37933

Table 2: *FFT Greedy Total Moves for Analysis 2:* Total number of moves computed using the greedy algorithm for different FFT DAGs under Red-Blue (RB) and Red-Orange-Blue (ROB) pebble game models, with varying processor counts. ROB uses 8× red pebbles for orange memory.

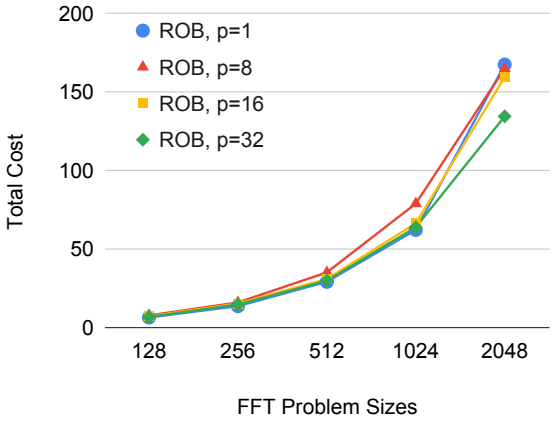

Figure 5: *ROB Total Cost - FFT (Analysis 2):* Total cost (scaled by a factor of 1000) for FFT DAGs under Red-Orange-Blue (ROB) and Red-Blue (RB) pebble games across varying processor counts keeping per-processor memory constant.

analysis on Per-Processor Memory Constant Setting because the minimum number of red pebbles for the resnet dags are too small (3 pebbles; Table 3), which is not plausible to divide among multiple processors to analyze the total memory constant setting.

5.2.1 ***Analysis 3: Keeping per-processor memory constant.*** In the RB setting (Figure 6), total cost increases steadily with model depth, with resnet152 incurring the highest cost. Increasing processor count from (p=1) to (p=32) provides moderate cost reductions, though improvements tend to saturate, especially for smaller models like resnet18 and resnet34. Notably, the cost curve for (p=32) displays irregular behavior at resnet34, indicating suboptimal reuse or scheduling effects under the greedy strategy. In contrast, the ROB model (Figure 7) demonstrates more consistent scaling with monotonically decreasing costs as processor count increases. With per-core memory fixed, ROB reduces 60–65 % of total cost for ResNet-18/34/50 across every processor count. For deeper backbones the relative gap narrows—roughly 38 % for ResNet-101 and 24 % for ResNet-152. This advantage of orange memory in enabling intermediate reuse and alleviating recomputation overhead is particularly evident in resnet152, where cost drops sharply from (p=1) to (p=32). Table 3 reports the total number of moves computed for fine-grained resnet DAGs under both models.

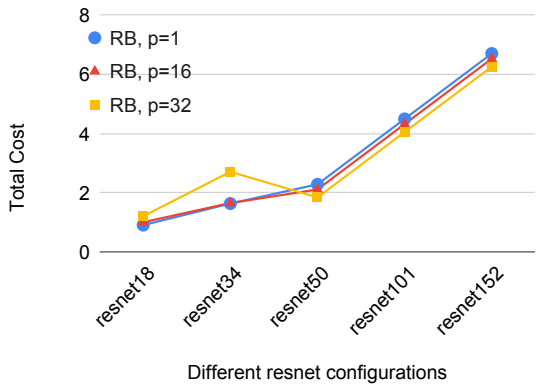

Figure 6: *RB Total Cost - resnet-coarse (Analysis 3):* Total cost (log-scaled by a factor of 1000) under the Red-Blue (RB) pebble game across coarse-grained resnet architectures and varying processor counts keeping per-processor memory constant.

6 SUMMARY AND FUTURE WORK

In this paper, we proposed pebble games as a powerful framework for modeling and designing efficient disaggregated memory systems. To address the unique challenges of multi-tenant cloud computing environments, we introduced a novel variant, the dynamic red-orange-blue pebble game with multiple processors (DyPeG), which captures the dynamic provisioning of memory resources in such scenarios. We evaluated DyPeG empirically using two representative applications, demonstrating its capability to effectively assess modern disaggregated memory systems. Our preliminary results highlight the potential of DyPeG as a robust tool for analyzing and optimizing memory architectures in cloud-based, multi-processor environments.

For future work, we plan to develop learning-based approaches to solve DyPeG and focus on simulating the concurrent execution of heterogeneous workloads across multiple processors, while

Models	nodes	edges	in_degree	upper bound	min_red	Single Processor		2 Processors		4 Processors		8 Processors		16 Processors		32 Processors	
						RB	ROB	RB	ROB	RB	ROB	RB	ROB	RB	ROB	RB	ROB
resnet18	71	78	2	8	3	140	140	141	141	141	141	146	146	169	169	192	192
resnet34	127	142	2	8	3	252	252	253	253	253	253	256	256	273	273	394	394
resnet50	177	192	2	8	3	352	352	353	353	353	353	352	352	349	349	336	336
resnet101	347	379	2	8	3	692	844	693	693	693	693	692	692	689	689	676	676
resnet152	517	566	2	8	3	1032	1354	1033	1033	1033	1033	1032	1032	1029	1029	1016	1016

Table 3: *Resnet Total Moves for Analysis 3:* Total number of moves computed using the greedy algorithm for different resnet DAGs under Red-Blue (RB) and Red-Orange-Blue (ROB) pebble game models, with varying processor counts. ROB uses 8× red pebbles for orange memory.

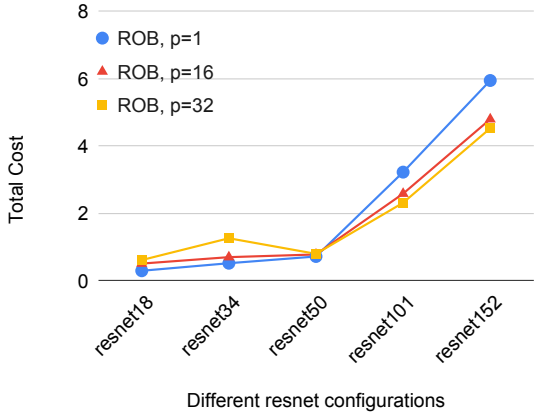

Figure 7: *ROB Total Cost - resnet-coarse (Analysis 3):* Total cost (log-scaled by a factor of 1000) under the Red-Orange-Blue (ROB) pebble game using 8× orange pebbles across coarse-grained resnet architectures.

maintaining compliance with Service Level Agreement (SLA) constraints. Furthermore, we want to explore the use of DyPeG to dynamically guide workload scheduling and memory allocation decisions on multi-tenant systems. We aim to enhance the scalability and real-world applicability of pebble games in complex and realistic distributed cloud computing scenarios.

ACKNOWLEDGMENTS

This work was supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 76125: "AMAIS - Advanced Memory to support Artificial Intelligence for Science". The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830, and by PRISM, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

REFERENCES

[1] Friedhelm Meyer auf der Heide. 1981. A comparison of two variations of a pebble game on graphs. *Theoretical Computer Science* 13, 3 (1981), 315–322.

[2] Toni Böhnlein, Pál András Papp, and Albert-Jan N Yzelman. 2025. Red-Blue Pebbling with Multiple Processors: Time, Communication and Memory Trade-Offs. In *International Colloquium on Structural Information and Communication Complexity*. Springer, 109–126.

[3] Timothy Carpenter, Fabrice Rastello, P Sadayappan, and Anastasios Sidiropoulos. 2016. Brief announcement: Approximating the i/o complexity of one-shot red-blue pebbling. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*. 161–163.

[4] Debendra Das Sharma, Robert Blankenship, and Daniel Berger. 2024. An Introduction to the Compute Express Link (CXL) Interconnect. *ACM Comput. Surv.* 56, 11, Article 290 (July 2024), 37 pages.

[5] Erik D. Demaine and Quanquan C. Liu. 2018. Red-Blue Pebble Game: Complexity of Computing the Trade-Off between Cache Size and Memory Transfers. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures* (Vienna, Austria) *(SPAA'18)*. Association for Computing Machinery, New York, NY, USA, 195–204. https://doi.org/10.1145/3210377.3210387

[6] Venmugil Elango, Fabrice Rastello, Louis-Noël Pouchet, Jagannathan Ramanujam, and Ponnuswamy Sadayappan. 2014. On characterizing the data movement complexity of computational DAGs for parallel execution. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*. 296–306.

[7] John R Gilbert, Thomas Lengauer, and Robert Endre Tarjan. 1979. The pebbling problem is complete in polynomial space. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*. 237–248.

[8] Niels Gleinig and Torsten Hoefler. 2022. The red-blue pebble game on trees and dags with large input. In *International Colloquium on Structural Information and Communication Complexity*. Springer, 135–153.

[9] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, and Thomas Moscibroda. 2020. Protean: VM Allocation Service at Scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 845–861.

[10] Pingyi Huo, Anusha Devulapally, Hasan Al Maruf, Minseo Park, Krishnakumar Nair, Meena Arunachalam, Gulsum Gudukbay Akbulut, Mahmut Taylan Kandemir, and Vijaykrishnan Narayanan. 2024. PIFS-Rec: Process-In-Fabric-Switch for Large-Scale Recommendation System Inferences. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 612–626.

[11] Hong Jia-Wei and Hsiang-Tsung Kung. 1981. I/O complexity: The red-blue pebble game. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*. 326–333.

[12] Grzegorz Kwasniewski, Marko Kabić, Maciej Besta, Joost VandeVondele, Raffaele Solcà, and Torsten Hoefler. 2019. Red-blue pebbling revisited: near optimal parallel matrix-matrix multiplication. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–22.

[13] Thomas Lengauer. 1981. Black-white pebbles and graph separation. *Acta Informatica* 16 (1981), 465–475.

[14] Quanquan Catherine Liu. 2017. *Red-blue and standard pebble games: Complexity and applications in the sequential and parallel models.* Ph. D. Dissertation. Massachusetts Institute of Technology.

[15] Pál András Papp and Roger Wattenhofer. 2020. On the hardness of red-blue pebble games. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*. 419–429.

[16] James L. Peterson. 1977. Petri Nets. *ACM Comput. Surv.* 9, 3 (Sept. 1977), 223–252. https://doi.org/10.1145/356698.356702

[17] Wolfgang Reisig. 2008. The Scholten/Dijkstra pebble game played straightly, distributedly, online and reversed. In *Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*. Springer, 589–595.

[18] John E Savage. 1995. Extending the Hong-Kung model to memory hierarchies. In *International Computing and Combinatorics Conference*. Springer, 270–281.

[19] Ravi Sethi. 1973. Complete register allocation problems *(STOC '73)*. Association for Computing Machinery, New York, NY, USA, 182–195. https://doi.org/10.1145/800125.804049

[20] Manuel Silva. 2012. 50 years after the PhD thesis of Carl Adam Petri: A perspective. *IFAC Proceedings Volumes* 45, 29 (2012), 13–20. 11th IFAC Workshop on Discrete Event Systems.

[21] Gajendra Pratap Singh. 2016. Applications of Petri nets in electrical, electronics and optimizations. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*. 2180–2184.

[22] Aleksandros Sobczyk. 2024. I/O complexity and pebble games with partial computations. *arXiv preprint arXiv:2410.22237* (2024).

[23] Xiaoyang Zhang, Junmin Xiao, and Guangming Tan. 2020. Communication Lower Bounds of Convolutions in CNNs. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '20)*. Association for Computing Machinery, New York, NY, USA, 591–593.