CAGE: A Contention-Aware Game-theoretic Model for Heterogeneous Resource Assignment

Diman Zad Tootaghaj, Farshid Farhat
The Pennsylvania State University
{*dxz149, fuf111*}*@cse.psu.edu*

***Abstract*—Traditional resource management systems rely on a centralized approach to manage users running on each resource. The centralized resource management system is not scalable for large-scale servers as the number of users running on shared resources is increasing dramatically and the centralized manager may not have enough information about applications' need. In this paper we propose a distributed game-theoretic resource management approach using market auction mechanism to find optimal strategy in a resource competition game. The applications learn through repeated interactions to choose their action on choosing the shared resources. Specifically, we look into two case studies of cache competition game and main processor and co-processor congestion game. We enforce costs for each resource and derive bidding strategy. Accurate evaluation of the proposed approach show that our distributed allocation is scalable and outperforms the static and traditional approaches.**

***Index Terms*—Game Theory; Contention; Resource sharing**

I. Introduction

The number of cores on chip multiporcessors (*CMP*) is increasing each year and it is believed that only many-core architectures can handle the massive parallel applications. Server-side *CMP*s usually have more than 16 cores and potentially more than hundreds of applications can run on each server. These systems are going to be the future generation of the multi-core processor servers. Applications running on these systems share the same resources like last level cache (*LLC*), interconnection network, memory controllers, off-chip memories, auxiliary processing capability like co-processors etc. Along with rapid growth of core integration, the performance of applications highly depend on the allocation of resources and specially the *contention* for shared resources [1, 2, 3, 4, 5, 6]. In particular, as the number of co-runners running on the shared resource increase, the magnitude of performance degradation increases. As a result, this new architectural paradigm introduces several new challenges in terms of scalability of resource management and assignment on these large-scale servers. Therefore, a scalable competition method between applications to reach the optimal assignment can significantly improve the performance of co-runners on a shared resource. Figure 1 shows an example of performance degradation for 10 *spec 2006* applications running on a shared 10MB *LLC* and solo run on 1MB *LLC*. Recently, many proposals target partitioning the cache space between applications such that (1) each application gets the minimum required space, so that per-application performance is guaranteed to be at an acceptable level, (2) system performance is improved by deciding how the remaining space should be allocated to each one. Prior schemes are marching towards these two goals, usually by trading off the system complexity and maximum system utilization. It is shown that neither a pure private *LLC*, nor a pure shared *LLC*, can provide optimal performance for different workloads [4]. In general Prior schemes have the following three challanges:

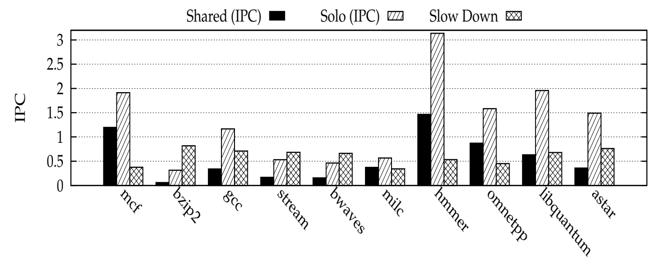

Fig. 1: Performance degradation of 10 different *spec 2006* applications sharing *LLC*.

1. Scalability: All of the prior schemes suffer from scalability; especially when the approach is tracking the application's dynamism [7].

2. Static-based: Most of the prior works, use static co-scheduling to degrade slow-down of co-running applications on the same shared cache. However, static-based approaches can not catch dynamic behavior of applications.

3. Fairness: Defining a single parameter for fairness is challenging for multiple applications, since applications have different performance benefits from each resource during each phase. In prior works fairness has been defined as a unique metric (eg. IPC, Power, Weighted Speed-up) for all applications. Therefore, in current approaches, the optimization goal of algorithms is the same for all applications. Consequently, we cannot sum up applications that desire different metrics in the same platform to decide on.

In this paper, we present a distributed heterogeneous resource assignment method to address all the above shortcomings including scalability, dynamism and fairness, while applications can get their desired performance based on their utility functions.

II. CAGE: A Market-based Contention-aware Game-theoretic resource assignment

Game theory has been used extensively in economics, political and mathematical decision making situations [8, 9, 10, 11].

Auction games are a class of games which has been used to formulate real world problems of assigning different resources between n users. Auction game framework can model resource competition, where the payoff (cost) of each application in the system is a function of the contention level (number of applications) in the game.

Inspired by nature predator-prey interactions in real life games, there exists a repeated interaction between competitors in a resource sharing game. We show that, assuming large number of applications the service rate of each application on each resource converges to each other. Furthermore, we show that the auction model is strategy-proof, such that no application can get more utilization by bidding more or less than the true value of the resource.

A. Model Description

Consider n applications and i instances of m different resources. Applications arrive in the system one at a time. The applications have to choose among m resources. There exists a bipartite graph between the matching of the applications and the resources.

We formulate our problem as an auction based mechanism to enforce cost/value updates for each resource as follows:

- **Valuation $\mathbf{v_{i,m}}$** : Any application has a valuation function which shows how much he benefits from ith resource. The valuation function at time $t = 0$ for cache contention case study is derived from the IPC (instruction per cycle) curves which is found using profiling, and for processor and co-processor contention case study is derived from the profiling solo performance metric of the application. However, in general, each application can choose its own utility function.
- **Observed information**: The observed information at each time step is the performance value of the selected action in the game. Therefore, the applications repeatedly update the history of their valuation function over time.
- **Belief updating**: At each iteration step of the auction, the applications update their valuation of each resource based on the observed performance on each resource. The update at time T is derived using the following formula:

$$v_{i,m}(T) = \frac{\sum_{t=0}^{T} \delta^{T-t} v_{i,m}(t)}{\sum_{t=0}^{T} \delta^{T-t}} \tag{1}$$

  Where $v_{i,m}(t)$ shows the observed valuation of resource m at time step t by user i in the system; δ shows the discount factor between 0 and 1 which shows how much a user relies on its past observations in the system. The discount factor is chosen to show the dynamics in the system. If the observed information in the system changes fast, the discount factor is nearly zero which means that we can't rely on the past observations very much. However if the system is more stable and the observed information does not change fast, the discount factor is chosen to be near 1. We choose the discount factor as the absolute value of the correlation coefficient of the observed values of the valuations at each iteration step which is calculated as follows:

$$\delta = \frac{E(v_{i,m})^2}{{\sigma_{v_{i,m}}}^2} \tag{2}$$

- **Action**: At each time step the applications decides which resource to bid and how much to bid for each resource.

B. Distributed Optimization Scheme

The goal is to design a repeated auction mechanism which is run by the operating system to guide the applications to choose their best resource allocation strategy. The applications' goal is to maximize their own performance and the operating system wants to maximize the total utility it gains from the applications. Then, each application can use its own utility function and evaluates the resources based on how much it likes that particular resource.

Applications' approach: The application i want to maximize the total utility with respect to a limited budget for all phase p of its execution time.

$$\begin{aligned} \forall i \in U \quad & maximize \quad \sum_{p=1}^{P_i} \sum_{m=1}^{M} v_{i,m,p} - b_{i,m,p}, \\ & subject\ to \quad \sum_{p=1}^{P_i} \sum_{m=1}^{M} b_{i,m,p} \leq B_i. \end{aligned} \tag{3}$$

OS's approach: The operating system wants to maximize the social welfare function which is translated into submitted bids from the applications in a limited resource constraints.

$$\begin{aligned} & maximize \quad \sum_{i=1}^{N} \sum_{p=1}^{P_i} \sum_{m=1}^{M} b_{i,m,p} A_{i,m,p}, \\ & subject\ to \quad \sum_{i=1}^{N} \sum_{m=1}^{M} A_{i,m,p} \leq A_{max}, \quad \forall p \in P, \\ & A_{i,m,p} \in \{0,1\}, \quad \forall i \in U, \ \forall m \in V, \ \forall p \in P. \end{aligned} \tag{4}$$

Definition 1: A strategy profile a is a pure Nash equilibrium if for every application i and every strategy $a'_i \neq a_i \in A$ we have $u_i(a_i, a_{-i}) \geq u_i(a'_i, a_{-i})$

Theorem 1: Suppose n risk-neutral applications whose valuations are derived uniformly and independently from the interval $[0, 1]$ compete for one resource which can be assigned to m application who have the highest bid in the auction. We will show that Bayes Nash equilibrium bidding strategy for each application in the system is to bid $\frac{n-m}{n-m+1} v_i$ whre v_i is the profit of application i for getting the specified resource.

Theorem 1, states that whenever there is a single resource that users compete to get it with different valuation functions, the Nash equilibrium strategy profile for risk-neutral users is to

Algorithm 1: CAGE: Parallel Auction for heterogeneous resource assignment.

```
Input: A bipartite Graph (U, V, E).
Output: The allocation of resources to applications.
1 At t=0 the valuation of each application for each
    resource is derived using profiling while running alone.
2 For each application $U_i \in U$, the first bottleneck resource
    is
        $$Bottleneck_{1,i} = V_{i,m} = arg \max_{m \in V}(v_{i,m} - p_m)$$
    Next, find the second bottleneck resource for each
    applications $U_i \in U$ in the system:
        $$Bottleneck_{2,i} = V_{i,k} = arg \max_{k \in V, k \neq m}(v_{i,k} - p_k)$$
3 Each application submits the bid for its first bottleneck
    resource using the following formula:
        $$b_{i,m} = V_m - V_k + p_j + \epsilon$$
    Each resource $V_j \in V$, which can be shared between $m$
    applications, is assigned to the $m$ highest bidding
    applications $Winner_j = i_1, i_2, ..., i_m$ and the price for
    that resource is updated as follows:
        $$p_j = arg \max_{i_1, i_2, ..., i_m \in U} \sum_{k=1}^{m}(b_{i_k,j})$$
4 The $minBid$ for each resource is updated as the
    minimum bid of $m$ applications who acquired the
    resource. That is
        $$minBid = arg \min_{i \in Winner_j}(b_{i,j})$$
```

bid $\frac{n-m}{n-m+1}v_i$. This term tends to the true value of the object when n is a large number.

In case of more than one resource competition we derive Algorithm 1 and will prove that it is Nash equilibrium in the game. The algorithm is inspired by work of Bertsekas [12] that uses an auction for network flow problems.

In the first step, all valuations are set to the solo-run of application's performance. Next, each application submits a bid for its first bottleneck resource. The bid should be larger than the price of the object which is intitialized to zero in the begining of the program. The applications only have incentive to bid a value no more than the difference of the first bottleneck and second bottleneck resource. Otherwise, it would submit a smaller bid to the second bottleneck and get the same revenue as paying more for the first bottleneck resource. In order to break the equal valuation function between two different applications, we use ϵ scaling such that at each iteration of the auction the prices should increase by a small number.

III. Case Studies

A. CPU Scale-up Scale-out Game

The experiment results of this section are run on *Stampede* cluster of *Texas Advanced Computing Center*. We executed *MiniGhost* application which is a part of *Mantevo* project [13] which uses difference stencils to solve partial differential equations using numerical methods. The applications use the profiling utility functions at $t = 0$ and during course of execution can update the utility function based on the observed performance on each core using Equation 1. Then, they can revisit their previous action on running the code on either the processor or co-processor during run-time.

Figure 2 shows the total execution time with respect to congestion we made in *Xeon* and *Xeon Phi*. In this experiment we ran the same problem size on a *Xeon* and *Xeon Phi* machine multiple times, so that we could see the effect of load on the total execution time of our application. It was observed that with the same number of threads *Xeon*'s performance degrades more than *Xeon phi*. Furthermore, it is shown that CAGE can bring in up to 106.6% improvement in total execution time of applications compared to static approach when the number of co-runners is six. The performance improvement would be significant when the number of co-runners increase. Figure 3 shows the performance comparison of CAGE and static approach which does not consider the congestion dynamism in the system and the decision is only made based on the parallelism level in the code.

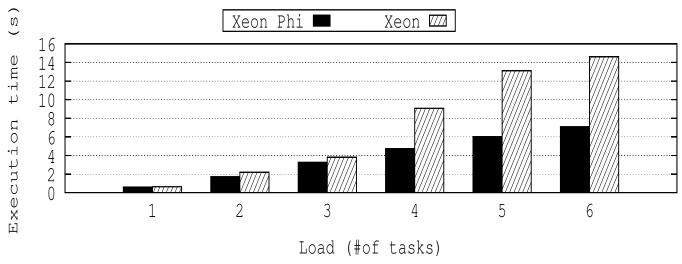

Fig. 2: Congestion effect on *Xeon* and *Xeon Phi* machines.

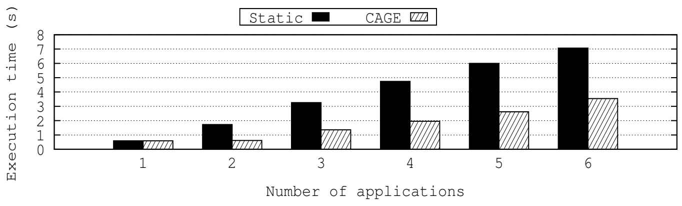

Fig. 3: Performance comparison of congestion-aware schedule versus static schedule.

B. A Case Study of Private and shared cache game

We use *Gem5* full system simulator in our experiment [14, 15]. Table I shows the experimental setup in our experiments.

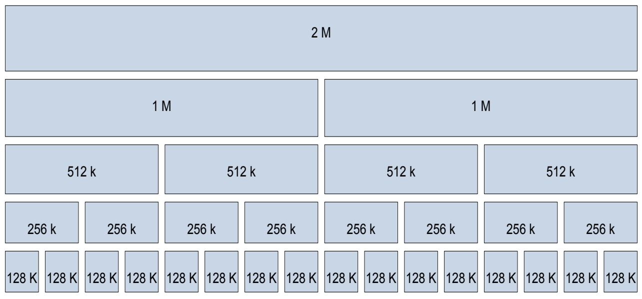

Fig. 4: Our proposed last level cache hierarchy model.

TABLE I: Experimental Setup.

Processors	Single threaded with private L1 instruction and data caches
Frequency	1GHz
L1 Private ICache	32 kB, 64-byte lines, 4-way associative
L1 Private DCache	32 kB, 64-byte lines, 4-way associative
L2 Shared Cache	128 kb-2 MB, 64-byte lines, 16-way associative
RAM	12 GB

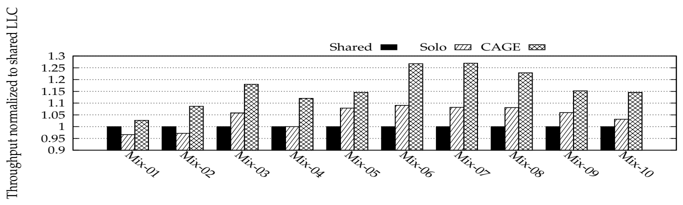

Fig. 5: Throughput of a shared, solo and CAGE cache allocation schemes.

To evaluate the performance of our proposed approach we use utility functions for different number of ways based on the applications' IPC for each cached size. These utility functions used at the start of the execution are found using either profiling techniques or stack distance profile [3] of applications assuming there is no co-runners in the system. Next, during run-time the applications can update their utility functions based on Equation 1. Therefore, there is a learning phase where applications learn about the state of the system and update the utilities accordingly.

Next, we use different mixes of 4 to 16 applications from *Spec 2006* to evaluate the performance of our proposed approach. Figure 5 shows the normalized throughput of 10 different mix of applications using CAGE, equal private cache partitions and completely shared cache space. Figure 6 shows the scalability of our proposed algorithm. When the number of co-runners increases from 2 to 16, the performance improves from 12.4% to 33.6% without any need to track each applications' performance in a central hardware.

IV. Conclusion

The paper proposes a distributed resource allocation mechanism for large scale servers. The traditional resource management system are not scalable, especially when tracking the application's dynamic behavior. The main cause of this

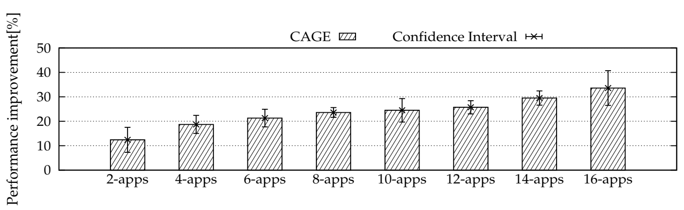

Fig. 6: Performance improvement of CAGE for different number of applications with respect to shared LLC for the case study of cache congestion game.

complexity is the centralized decision making which leads to higher time and space complexity. With increasing number of cores per chip, the scalability of assigning different resources to different applications becomes more challenging in future generation CMP systems. In addition, diversity in application's need make a single objective function inefficient to get an optimal and fair performance metric.

We introduce a framework to map the allocation problem to the known auction economy model where the application get virtual money and based on the utility metric they compete for the shared resource.

References

[1] L. Tang et al. The impact of memory subsystem resource sharing on datacenter applications. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, 2011.

[2] S. Zhuravlev et al. Addressing shared resource contention in multicore processors via scheduling. In *ACM SIGARCH Computer Architecture News*. ACM, 2010.

[3] S. Kim et al. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*. IEEE Computer Society, 2004.

[4] S. Cho et al. Managing distributed, shared l2 caches through os-level page allocation. In *MICRO*, 2006.

[5] F. Farhat et al. Stochastic modeling and optimization of stragglers. *IEEE Transactions on Cloud Computing*, 2016.

[6] D. Z. Tootaghaj et al. Optimal placement of cores, caches and memory controllers in network on-chip. *arXiv preprint arXiv:1607.04298*, 2016.

[7] M. K. Qureshi et al. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *MICRO*. IEEE Computer Society, 2006.

[8] D. Z. Tootaghaj et al. Game-theoretic approach to mitigate packet dropping in wireless ad-hoc networks. In *IEEE CCNC*, 2011.

[9] Kotobi et al. Spectrum sharing via hybrid cognitive players evaluated by an m/d/1 queuing model. *EURASIP Journal on Wireless Communications and Networking*, 2017.

[10] K. Kotobi et al. Introduction of vigilante players in cognitive networks with moving greedy players. In *Vehicular Technology Conference (VTC Fall)*. IEEE, 2015.

[11] D. Z. Tootaghaj et al. Risk of attack coefficient effect on availability of ad-hoc networks. In *IEEE CCNC*, 2011.

[12] D. P. Bertsekas. *Network Optimization: continuous and discrete methods*. Athena Scientific, 1998.

[13] `http:/manetovo.org`.

[14] N. Binkert et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 2011.

[15] `http://gem5.org/`.