AMTH 31 Final Project: Sigmoidal Programming for Optimization of Cache Allocation On the MIND Distributed Computer

Anthony Jiang ANTHONY.JIANG@YALE.EDU
Department of Computer Science
Yale University
New Haven, CT 06511, USA

Editor:

1. Introduction

Disaggregated Memory Architecture is the approach of physically separating Memory and CPU resources into distinct "blades" inside a data center connected over a high-performance network fabric. In this way, instead of connecting numerous self-contained computing nodes with a network fabric, the network fabric connects dedicated compute, memory, and storage resources. This architecture offers the ability to more elastically scale the system's components individually to improve utilization, enable hardware heterogeneity, and reinforce fault tolerance.

Existing disaggregated memory architectures face several pressing challenges. First, it is difficult to reduce the latency for remote memory access to rival that of a simple internal bus connection in a traditional chip. Second, while disaggregated memory is theoretically more amenable to elastic scaling, this fully elastic scaling has yet to be implemented. Finally, existing programs typically require modification to function on disaggregated computers.

MIND offers a solution to all three by migrating memory management into the network fabric. In this way, the network most closely behaves like the CPU-memory interconnect in a traditional computer. This solution offers a natural "virtualization" of traditional architectures, but moving this data onto the network fabric poses its own host of issues relating to the distributed nature of network fabric logic and the physical restrictions on existing switch architectures.

2. Problem

(Primarily described in a paper from Khandelwal lab) MIND targets datacenter deployments, where multi-tenancy (running multiple containers or VMs on a single physical computer) is a common requirement. However, multi-tenancy requires performance isolation for the shared resources on the computer. This reintroduces the multi-capacity bin packing problem that resource disaggregation is supposed to solve. Multi-tenancy introduces another problem of performance isolation and fairness in a rack-scale. The traditional abstraction such as virtual machines (VMs) or containers provides performance isolation and

fairness in a server granularity. However, in resource disaggregation, user can run their container spanning across multiple resource blades expecting performance isolation and fairness between containers of different users. For example, a workload can be running on four compute blades, two memory blades, and one storage blade, while all the resource blades are shared with other workloads. Thus, the performance isolation in a disaggregated rack should mutually consider other types of resources such as CPU and memory.

Usage of local memory (behaving as cache) and network resources (behaving as bus capacity) is not known a priori and fluctuates in runtime.

2.1 Problem Formulation

We let the total of Compute Blade memory (behaving as cache) equal C. That is, for n compute blades each with r_i local DRAM, $C = \sum_{i=1}^{n} r_i$. Further, we let the total network bandwidth be B. Network bandwidth is a linear function of remote RAM, so this functions as a proxy for remote RAM.

We assign to every container on the system c_i local memory and b_i bandwidth (remote memory). The working set for the process is located either in local memory or remote memory, so $c_i + b_i$ must equal the demand for the process. Increasing c_i permits a reduction in b_i, while a decrease in c_i requires an increase in b_i. Let $f_i : \mathbb{R}^2 \mapsto \mathbb{R}$ represent the number of memory accesses servable in unit time for a combination of (b, c).

As of now, we only have data on the relation between cache size and throughput, so we will be modelling univariate relationships between a single resource allocation and total throughput.

2.1.1 Characteristics of f

We have two observations about f.

- f_i is increasing in c_i, so the first derivative is positive in c_i and b_i up until a certain point.
- f_i experiences diminishing returns past a certain point

Empirically, we observe that f for various functions seems be approximated with sigmoid functions (based on limited data points). (Figure 1).

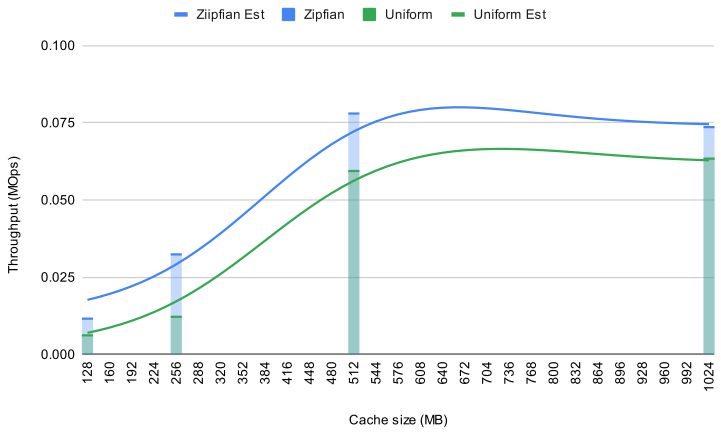

Figure 1: Cache and Throughput for Zipfian and Uniform RocksDB Workloads

based on the data collected on a limited set of sample programs. These are roughly approximated by hand as follows:

$$f_{\text{rocksDB on zipfian data}}(x) = 0.012 + \frac{0.075}{1+\exp\left(-0.015(x-384)\right)} - \frac{0.01}{1+\exp(-0.01(x-800))}$$

$$f_{\text{rocksDB on uniform data}}(x) = 0.002 + \frac{0.065}{1+\exp\left(-0.01(x-384)\right)} - \frac{0.018}{1+\exp(-0.01(x-700))}$$

where x is the allocated cache size in MB. For sake of experiment, we can consider varying each parameter (the numerator L, the y-intercept C, the coefficient of x which we term c_1, and the "midpoint" of the sigmoid m).

In these specific instances, data was only collected for cache sizzes of 128, 256, 512, and 128 MB. This does indeed align with the conclusion drawn by Ghodsi et. al.

Thus, we have the following optimization problem:

2.1.2 Problem Formulation

$$\max \sum_{i=1}^{n} f_i(c_i)$$

$$\text{subject to } \sum_{i=1}^{n} c_i = C$$

where f_i is sigmoidal,

2.2 Current Optimization technique

There are currently four optimization objectives that are of interest right now

- Sharing incentive: $f_i(c_i, b_i) \geq f_i(C/n, B/n) \forall i$

- Envy-freeness: A container should not be able to improve it's performance by choosing another container's allocation: $f_i(c_i, b_i) \geq f_i(c_j, b_j) \forall i, j$
- Strategy-proofness: Containers should not be able to increase their performance by lying about their performance
- Pareto-efficiency

The current approach to meeting all four objectives is a trading-based algorithm inspired by Ghodsi et. al. where individual threads declare their demands and the derivative of their throughput offered by adding a δ to the cache, then determining whether threads would be willing to trade cache for memory (bandwidth).

However, this project will evaluate the feasibility of a direct computation of the optimal resource distribution using sigmoidal programming as defined by Udell and Boyd.

2.3 Sigmoidal Programming

Udell and Boyd in a 2013 paper introduce the application of convex optimization techniques to sigmoidal programming. They define sigmoidal programming as optimizations of the form:

$$\max \sum_{i=1}^{n} f_i(x_i)$$
$$\text{subject to } x \in C$$

where f constitute sigmoind functions, C is a nonempty bounded closed convex set. A prime example of sigmoid functions include the logistic function $f(x) = \frac{1}{1+\exp -x}$.

The technique proposed by Udell and Boyd is inspired by the branch and bound method. This is an interative method to divide the search space into regions (branch), then bound each of them to evaluate which regions are worth investigating further. Like the trading algorithm, this is an iterative approach that will converge to the desired solution. Udell and Boyd describe the algorithm in figure 2.

The algorithm begins with an initial rectangular region that includes C, call it Q. Thus, the optimization can be written as:

$$\max \sum_{i=1}^{n} f_i(x_i)$$
$$\text{subject to } x \in C \cap Q$$

Let the optimal value for this be $p^*(Q)$. This region is divided the initial region into subregions, and each subregion is evaluated for which one has the suitable upper bound to include the optimal value of x, or $\hat{x}$. Each subregion is then bounded to evaluate whether the optimal can be found in it. To be precise, the upper bound and the lower bound on this region are evaluated by solving the following optimization problem:

Algorithm 1 Branch and bound

given initial rectangle Q_0, tolerance ϵ
compute lower bound $L(Q_0)$ and upper bound $U(Q_0)$ on Q_0
initialize global lower bound LB= $L(Q_0)$, global upper bound UB= $U(Q_0)$, partition $\mathcal{Q} = \{Q_0\}$
while UB - LB $> \epsilon$ **do**
  select $Q \in \mathcal{Q}$ with highest upper bound $U(Q)$
  update global upper bound UB $\leftarrow U(Q)$
  split Q into left and right subrectangles Q_L and Q_R
  compute lower bound $L(Q_L)$ and upper bound $U(Q_L)$ on Q_L
  compute lower bound $L(Q_R)$ and upper bound $U(Q_R)$ on Q_R
  update global lower bound LB $\leftarrow \max(\text{LB}, L(Q_L), L(Q_R))$
  update partition $\mathcal{Q} = \mathcal{Q} \cup \{Q_L, Q_R\} \setminus \{Q\}$
end while

Figure 2: Branch and Bound algorithm in Udell and Boyd

$$\max \sum_{i=1}^{n} \hat{f}_i(x_i)$$
$$\text{subject to } x \in C \cap Q \tag{2}$$

where $f\hat{x}$ is the "concave envelope" of f, or the smallest concave function greater than or equal to the sigmoid. This the optimal value of this maximization function be formally $U(Q)$ and let the x value that creates this be x^*. Udell and Boyd formally define this envelope as:

$$\hat{f(x)} = \begin{cases} f(l) + \frac{f(w)-f(l)}{w-l}(x-l)l & \leq x \leq w \\ f(x) & w \leq x \leq u \end{cases}$$

where w is the x component of the highest point from which a tangent can be drawn through the origin point. Udell and Boyd give the illustration (figure 3):

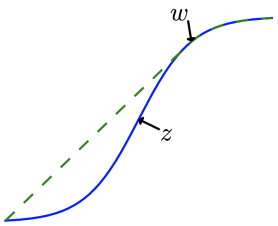

Figure 3: Illustration of a convex envelope from Udell and Boyd

With this function, we know that $U(Q) = \sum \hat{f}(x^*(Q)) \geq p^*(Q)$ because $\hat{f}(x)$ is at least as large as $f(x)$. Likewise, we know that $p^*(!) \geq \sum f(x^*(Q))$ because $\sum f(x^*(Q))$ is attainable, so $p^*(x)$ must be at least as large as it.

With this convex hull, we can bound:

$$U(Q) \geq p^*(Q) \geq L(Q)$$

Knowing this, we can iteratively evaluate all of the subregions for suitability, and then eventually converge on not just the optimal value, but also the optimal region. This is all implemented in the julia package `SigmoidalProgramming`.

2.4 Simulation

2.4.1 Function approximation

Given the lack of physical data, for the purposes of this project, we must generate dummy data. For the scope of this project, we are using data centered around the data points already observed.

We simulate the workloads as:

$$f(x) = C + \frac{L_1}{1 + \exp(-c_1(x - c_2))} - \frac{L_2}{1 + \exp(-c_{1,1}(x - c_{2,2}))}$$

Moreover, we implement our optimization using `SigmoidalProgramming`. With this, we implement the difference of logistic functions as following.

```
function gen_log_funct_2(C, L, c1, c2, C2, L2, c12, c22)
    function(x)
        C + L/(1 + exp(-c1 * (x - c2)))  - \\
        L2/(1 + exp(-c12* (x - c22)))
    end
end
```

```
function gen_log_funct_prime_2(C, L, c1, c2, C2, L2, c12, c22)
    function(x)
        (c1 * L * exp(-c1 * (x - c2)))/(1 + exp(-c1 * (x - c2)))^2 - \\
        (c12 * L2 * exp(-c12 * (x - c22)) )/(1 + exp(-c12 * (x - c22)))^2
    end
end
```

The second function, `gen_log_funct_prime_2` is required because the library requires the derivative be included in order to conduct the optimization. ‘

2.4.2 Simulation results

The following simulated throughput-cache relations: were generated. These can be graphed as:

With this, the following optimization was computed:

for a total throughput of 0.196, assuming a no-starvation lower bound of 128 MB for every process.

C	L_1	c_1	c_2	L_2	$c_{1,2}$	$c_{2,2}$
0.00067309	0.066790	0.0081430	399.06	0.006584	0.0146534	718.00
0.0020695	0.062549	0.0110708	497.8	0.0076472	0.0148187	794.82
0.000441819	0.066520	0.0069228	469.245	0.0087363	0.00750344	727.09
0.0045581	0.067632	0.0126916	343.61	0.0057252	0.0072472	830.40
0.0049620	0.062903	0.0137739	461.266	0.0077767	0.0062636	854.87)]

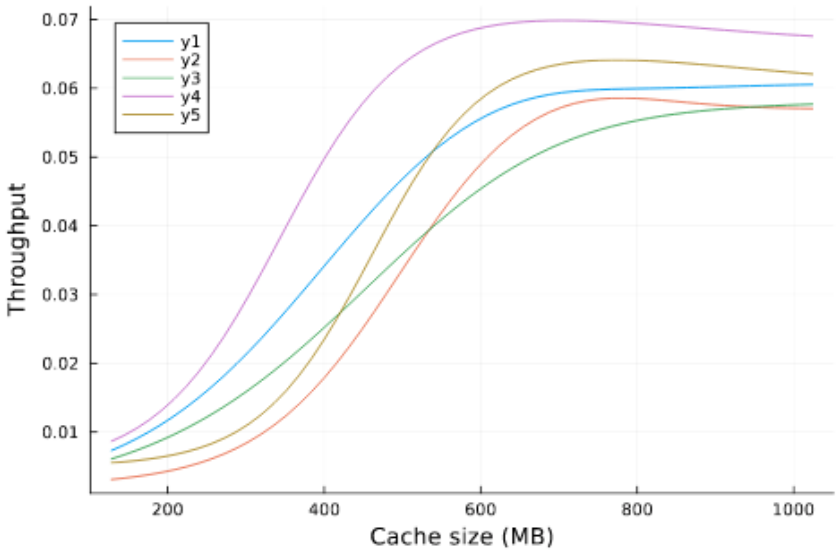

Figure 4: Simulated Workload Cache and Throughput

x_1	x_2	x_3	x_4	x_5
608.7	128.0	144.97	523.8	642.4

2.5 Conclusion

Sigmoidal programming offers a potential solution to the problem of allocating cache resources for a distributed computer,and this paper has validated the potential for this approach in efficiently computing allocations without an iterative trading algorithm. In short, this functions as a "reference" point for the trading algorithm, underscoring the value in having complete knowledge while conducting optimization instead of relying on online techniques.