ILLINOIS TECH

College of Computing

Simulating a Coalescent Computing Environment with Minimal Overhead

MD ALI

MALI54@HAWK.IIT.EDU

CS 546: PARALLEL AND DISTRIBUTED PROCESSING

---

1 Introduction

With the dynamic change taking place in edge computing components the line between local and remote has grown ever closer. This means that under the user end that they will ultimately have endless computational capabilities that is achieved by *coalescent computing* [4]. It is important to model the timing of such a system under different metrics where the computational power is flexible and not fixed. With this in mind we propose simulating the components using an x86 multi-core simulator *Sniper* [2] that is able to simulate a coalescent computing environment and measuring the number of loads from remote cache.

This brings up the notion of a distributed operating system or a *Single System Image (SSI)* [1] where these heterogeneous components and the overall distributed operating system is hidden. The available resources that are presented by a coalescent computing environment should be presented to the users as a single computing resource even though in the background the environment is borrowing computational resources from several cores in our case. With our proposal of simulating the coalescent computing environment, we would also like to achieve a scaleable and distributed operating system to measure the remote cache line transfers.

2 Background Information

The idea of coalescent computing can be summarized as *"Users' devices experience a coalescence of resources proportional to proximity as users move through the physical environment."* [4] This makes that the distributed environment would need to have a fast network components through either Bluetooth or Wi-Fi capabilities. This further develops the need to have a reliable, fast network within our simulation for a distributed system with remote cache lines.

With Distributed Shared Memory (DSM) being used more such as *Concordia* [9], where they proposed a fast in-network cache coherence which are backed by programmable switches. This further develops the idea that we will need to simulate a fast network connection for coalescent computing to be able to utilize remote caches and utilizing a in-network cache coherence protocol in a distributed system would be ideal. This will ultimately alleviate the cache coherent issues with remote caching since the overarching of a coalescent environment should have fast networks to be able to support an in-network cache coherence.

Scale-Out NUMA [6] is another distributed system but instead of using in-network cache coherence, it utilizes distributed in-memory processing. This relies on a remote memory controller that goes into the node's local coherence hierarchy. We will need to leverage this concept as well measure the remote cache transfers as through

configuration it might alleviate some more overhead as many of these remote cache loads will be distributed across many machines/cores.

Going off Scale-Out NUMA, that comes to another study that we must leverage to limit overhead in the simulation which is *reuse distance analysis* [7] which is a very well known tool that predicts cache performance. In out case, we would like to know the estimated timing for a remote cache performance in a coalescent computing environment.

Lastly, we would need to take in account for a *latency-tolerant system* [5] that will have to be mostly simulated in software. The most important parts are to have concurrency in global memory reference within the coalescent environment, have lightweight multi-threading available in the simulation, and overall create a low-overhead synchronization. In regards to concurrency in global memory reference we have to fully utilize the bandwidth. With lightweight multi-threading, we expect to needs a dynamic system that can support hundreds of threads to tolerate a cluster's network latency so we must implement in the simulation some low overhead. With that being said, we must also have low-overhead where the operations of each component in the environment does not block the processor's pipeline, so that other threads can proceed.

3 Problem Statement

The problem that arises is that we must simulate these key factors before we deploy them to get some idea about how the remote cache transfers would behave in a coalescent computing environment, as well as a reasonable amount of resources for any given node that is entering and leaving the distributed system. This would involve with simulation but also an accurate prediction of remote caches that need to calculated in real time, this would involve leveraging the tools and existing methods from previous studies.

4 Proposed Solution

We aim to utilize *Sniper* [2], which is a parallel x86 multi-core simulator that utilizes the Graphite simulation infrastructure. Here we will be using Jebe, a 64-core server, that will be running various benchmarks from the NAS benchmarks. The simulation will consists a user that would want to offload some heavy computation (NAS benchmarks, classes A-C) that their device will not be able to handle. The simulation will then showcase shared memory which will happen over a cache coherence network, which will project the performance of the coalscent environment. Utilizing the NAS

benchmarks on the simulator to get data on the remote cache line transfers and then apply them in a distributed environment in the constraints of the configuration files. We will ultimately use these measurements to be able to simulate and predicate an accurate measurement for a dynamic coalescent computing environment.

5 Evaluation

Our evaluation will consists of comparing the NAS benchmark data varying from different size classes, different computation e.g. Fourier transform, conjugate gradient, multi-grid, etc. and then ultimately offloading the computation to a remote cache lines. This will emulate a distributed system that we can compare to a normal distributed server-client simulation to show an increase in computation power while keeping the simulated cost of remote cache line transfers to a minimum utilizing previous methods.

6 Conclusions

We expect to see that our simulation is able to run faster, even with the latency of having remote caches. This will ultimately compare to your typical server-client simulation to show case that the computation power is worth the small amount of latency that will be used in the remote caches that are simulated. This should showcase that a coalescent computing environment should have no issues being implemented in a way where the latency shouldn't even be noticed by a user even with heterogeneous components in the distributed environment.

7 Timeline

Week	Goal
1	Read papers and compile the relevant information
2	Draw out design overhead
3	Code simulation configuration in Sniper
4	Present draft of completion
5	Fix any experimental mishalfs
6	Recreate experimental evaluation
7	Complete final report
8	Complete final presentation

References

[1] Rajkumar Buyya, Toni Cortes, and Hai Jin. "Single System Image". In: *IJHPCA* 15 (June 2001), pp. 124–135. DOI: 10.1177/109434200101500205.

[2] Trevor E. Carlson et al. "An Evaluation of High-Level Mechanistic Core Models". In: *ACM Transactions on Architecture and Code Optimization (TACO)* (2014). ISSN: 1544-3566. DOI: 10.1145/2629677.

[3] A. Gupta and W.-D. Weber. "Cache invalidation patterns in shared-memory multiprocessors". In: *IEEE Transactions on Computers* 41.7 (1992), pp. 794–810. DOI: 10.1109/12.256449.

[4] Kyle Hale. "Coalescent Computing". In: *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems*. APSys '21. Hong Kong, China: Association for Computing Machinery, 2021, pp. 79–88. DOI: 10.1145/3476886.3477503. URL: https://doi.org/10.1145/3476886.3477503.

[5] Jacob Nelson et al. "Crunching Large Graphs with Commodity Processors". In: *3rd USENIX Workshop on Hot Topics in Parallelism (HotPar 11)*. Berkeley, CA: USENIX Association, May 2011. URL: https://www.usenix.org/conference/hotpar11/crunching-large-graphs-commodity-processors.

[6] Stanko Novakovic et al. "Scale-out NUMA". In: *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '14. Salt Lake City, Utah, USA: Association for Computing Machinery, 2014, 3–18. ISBN: 9781450323055. DOI: 10.1145/2541940.2541965. URL: https://doi.org/10.1145/2541940.2541965.

[7] Derek Schuff, Milind Kulkarni, and Vijay Pai. "Accelerating Multicore Reuse Distance Analysis with Sampling and Parallelization". In: Nov. 2010, pp. 53–64. DOI: 10.1145/1854273.1854286.

[8] Ya-Yunn Su and Jason Flinn. "Slingshot: Deploying Stateful Services in Wireless Hotspots". In: *Third International Conference on Mobile Systems, Applications, and Services (MobiSys2005)*. Seattle, WA: USENIX Association, June 2005. URL: https://www.usenix.org/conference/mobisys2005/slingshot-deploying-stateful-services-wireless-hotspots.

[9] Qing Wang et al. "Concordia: Distributed Shared Memory with In-Network Cache Coherence". In: *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, Feb. 2021, pp. 277–292. ISBN: 978-1-939133-20-5. URL: https://www.usenix.org/conference/fast21/presentation/wang.