Disk-Locality in Datacenter Computing Considered Irrelevant

Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, Ion Stoica
University of California, Berkeley
{*ganesha, alig, shenker, istoica*}*@cs.berkeley.edu*

1 Introduction

Data center computing is becoming pervasive in many organizations. Computing frameworks such as MapReduce [17], Hadoop [6] and Dryad [25], split jobs into small tasks that are run on the cluster's compute nodes. Through these frameworks, computation can be performed on large datasets in a fault-tolerant way, while hiding the complexities of the distributed nature of the cluster. For these reasons, a considerable work has been done to improve the efficiency of these frameworks.

A popular approach to improve the efficiency of cluster computing frameworks has been to increase *disk-locality* – the fraction of tasks that run on nodes that have the task's input data stored on local disk. Several techniques have been proposed to increase disk-locality, including Delay-scheduling [26] and Scarlett [18]. In many systems, disk-locality is one of the main performance metrics used to evaluate efficiency [15, 17, 25]. Some systems, such as Quincy [21], go as far as defining fairness in terms of disk-locality, preferring to evict tasks to ensure fair distribution of disk-locality across jobs.

This paper takes the position that *disk-locality is going to be irrelevant in cluster computing*, and considers the implications this will have on datacenter computing research. The quest for disk-locality is based on two assumptions: (a) disk bandwidths exceed network bandwidths, (b) disk I/O constitutes a considerable fraction of a task's lifetime. We show that current trends undermine both these assumptions.

As observed many times in the past, networking technology is improving at a much faster pace than disk speeds [9]. We confirm these results, showing that – for a typical hardware setup – reading from local disk is only about 8% faster than reading from the disk of another node in the same rack, results which are consistent with the ones published elsewhere [23]. As for communication across racks, a clear trend is towards full-bisection topologies [24], which eliminate the bandwidth over-subscription. Such topologies have already been adopted in several datacenters [12]. This ensures that bandwidth across racks will be equal to the bandwidth within a rack.

Another trend that strengthens our thesis is the need to save more and more data in clusters. The need for storage space outweighs affordable storage, and the gap is projected to continue to expand [2]. The ever-increasing demand for storage not only makes solid state devices (SSDs) economically infeasible to deploy as a primary storage medium [5, 9, 16], but more importantly has led to the use of data compression. Compression significantly decreases the size of the data read from the disk, which correspondingly reduces the fraction of I/O in a task's lifetime. The net result is that compression further decreases the need for disk-locality. Indeed, our analysis of logs from Facebook show that disk-locality results in little, if any, improvement of task lengths.

While the above trends imply that tasks whose inputs are stored on disk can run *anywhere*, this is not the case for tasks whose inputs are in memory. Indeed, reading from local memory is two orders of magnitude faster than reading from local disk as well as reading from a remote host's disk or memory. The advent of full bisection bandwidth and optical networks are unlikely to have a material impact on this discrepancy, as inter-host latency still remains an insurmountable barrier. Thus, we expect *memory-locality* to play an ever-important role in the design of cluster computing frameworks.

While faster memory reads presents an opportunity to significantly speed-up jobs, it comes with its own set of challenges. In particular, there is at least two orders of magnitude discrepancy between the disk and memory capacities in today's clusters. Thus, replacing disks with memory is not feasible, which leaves us with one natural option: use memory as a *cache* to improve job completion times. This constitutes a significant difference from the RAMCloud [22] vision in that only a fraction of the data will be (and can be) in memory at any given time.

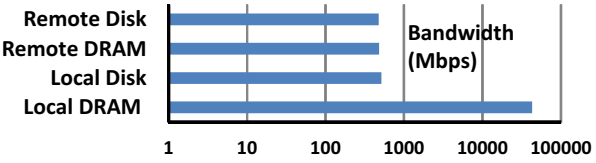
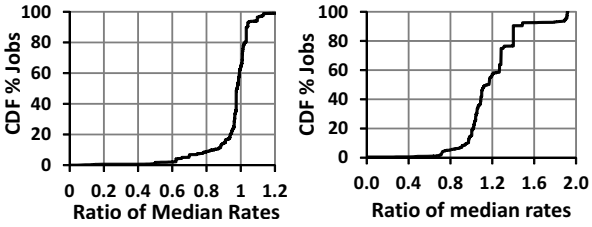


Figure 1: **Comparison of disk and memory read bandwidths locally and across the network from microbenchmark experiments.**



(a) Rack-local/Node-local (b) Rack-local/Off-rack

Figure 2: **(a) Rack-local tasks progress at the same rate as node-local tasks. (b) Off-rack tasks are only 1.2x slower at median despite an over-subscription of 10.**

In-memory caching however does not come without challenges. First, memory is not large enough to fit the inputs of all jobs. In our Facebook traces, the aggregate size of job inputs is an order of magnitude more than the available memory. Second, as many as 75% of blocks are accessed only once, which reduces the effectiveness of caching. Finally, to ensure good performance, we would need to cache all of the job's inputs. Even if a single task reads its input from the disk, it may become an outlier, which prolongs the job completion time.

Surprisingly, despite these challenges, we show that as many as 64% of all jobs achieve memory locality for *all* their tasks when using the LFU replacement policy. This is in large part due to the heavy-tailed nature of the workload, *i.e.,* the majority of jobs access only a small fraction of the blocks. Even more surprisingly, the inputs of the vast majority (96%) of active jobs *can* fit into a fraction of the cluster's total memory. This suggests there is a large potential for further improving most job completion times by developing new cache replacement and pre-fetching schemes.

Going forward, we solicit research on several problems related to that of providing an in-memory cache for datacenters. First, our analysis of Hadoop logs from Facebook show that a large fraction of the data is only accessed once, calling for techniques to prefetch data. Second, while traditional cache replacement techniques like LFU provide encouraging memory-locality, there is an opportunity for designing replacement schemes that minimize job-completion time, as opposed to bluntly increasing memory-locality. Finally, the rapid churn of file blocks in the caches will pose a challenge to servers that store metadata about block locations [28]. A scalable architecture is therefore required.

2 Disk-Locality Considered Irrelevant

We start by highlighting recent advances in networking. Thereafter we look at storage and data trends and finally see how SSDs will affect our claims.

2.1 Fast Networks

Advances in networking technology have seen link speeds in switches and server NICs increasing considerably. Switches with aggregate link speeds of 40Gbps and 100Gbps are commercially available [1]. Even server NIC rates of 10Gps and 25Gbps are expected to be popular in a couple of years [9].

Thus, as observed previously, reads from local disk are comparable to reads from local network. Figure 1 confirms this by showing that the bandwidth difference is about 8%[1]. This is consistent with bandwidth numbers in a recent Google report [23] over 2,000 servers.

The main motivation for disk-locality has, however, mostly been driven by off-rack bandwidth oversubscription [17]. Network topologies proposed in recent research significantly reduce or eliminate these over-subscription ratios [24]. These clos-based topologies have demonstrated full bisection bandwidth even when all the servers communicate with each other. In addition to providing full bisection bandwidth, their use of commodity switches enable remarkable cost savings for larger datacenters [24]. For these reasons, they have already been adopted by several datacenters [12], and this trend is only likely to continue.

Deployment Validation Hadoop tasks running at Facebook's datacenter corroborate our thesis. This datacenter consisted of over 3,000 machines, networked using the traditional three-tiered topology, and mostly ran data-intensive jobs, which spent the major fraction of their lifetime reading input data. To measure the effect of locality on task durations, we denote a task's *progress rate* as $\frac{(data\_read+data\_written)}{duration}$. We use the progress rate of a task as opposed to its duration to be immune to imbalance among tasks within a job in the amount of data they read/write. We compared the progress rates of node-local and rack-local tasks for a week in October 2010. For each job, we took the ratio between the median of its progress rates for rack-local and node-local tasks. Figure 2a plots the CDF for these ratios. We observe that 85% of the jobs have this ratio centered around 1.0 (in the window of 0.9 to 1.1). Note that only 4% of jobs have it

[1]The experiments were done on the DETER [4] testbed that have 1Gbps network links.

less than 0.7 (rack-local tasks being appreciably slower), essentially indicating that vying for node-locality does not buy much over rack-locality. This is explained by the above observation that disk read bandwidths are comparable to rack-local bandwidths.

What if we can stripe data from multiple disks at a time? This is a reasonable way to increase disk read throughputs as they have plateaued stand-alone. However, to compare with imminent network speeds of 10 and 25Gbps, we will need to connect up to 50 disks per server. In addition to raising scalability questions of the central disk controller, this is also practically challenging given the physical space constaints in a rack.

2.2 Storage Crunch

A long standing assumption in data-intensive computing has been that read durations dominate the lifetime of tasks. For such I/O-intensive tasks, naturally, speeding-up reads has significant impact on reducing task completion times. However, a recent phenomenon in datacenters challenge this. Under pressure to cope with the ever-growing volumes of data, datacenters have resorted to compression and de-replication of data. We first describe the reasons behind this storage crunch before proceeding to explain why shooting for data-locality is an unattractive option in this scenario.

Efficacy of data mining algorithms for online recommendation services (e.g., context-based advertisements), which form the main source of revenue for modern Internet services, improves with increase in the quantity of data. This has placed the storage infrastructure under strain (e.g., at Facebook [13] and Microsoft). While servers are stacked with multiple disks (typically, 2TB each up to a total of 12TB), they are limited by practical aspects like the physical space available on the rack. There is also a limit to the number of servers that can be deployed in a datacenter. Increasing them beyond a point has non-linear costs, i.e., migrating to a new datacenter with additional costs.

Compression Facebook datacenters use a number of techniques to deal with the storage crunch. Data is stored compressed. Since the primary driver of datacenter jobs are large text data blobs of structured data (e.g., web crawls) [3], they can be highly compressed. Tasks now read compressed data and decompress it before processing. High compression ratios mean significantly less data to read thereby obviating the need for disk-locality.

Hadoop task durations from Facebook, where data is compressed with gzip, support this observation. Figure 2b captures the median progress ratio between tasks executing rack-locally to off-rack. We expected off-rack tasks to be significantly slower as the network was oversubscribed by ten. On the contrary, running a task off-rack turns out to be only 1.2x and 1.4x slower compared to rack-local execution, at median and 95^{th} percentile. We consider this to be a direct effect of compression.

De-replication Aged data has lower number of replicas (as low as one copy) compared to other data. With just one copy, even under moderate cluster utilization, the probability of finding a disk-local slot is nearly negligible, making it an unattractive target to vie for. Analysis of Hadoop jobs from Facebook underscores the difficulty in attaining disk-locality: overall, only 34% of tasks run on the same node that has the input data.

Therefore, we conclude that techniques to deal with the storage crunch, in addition to network speeds improving vis-a-vis disk bandwidths, make data-locality irrelevant in datacenter computing.

2.3 Why SSDs will not help

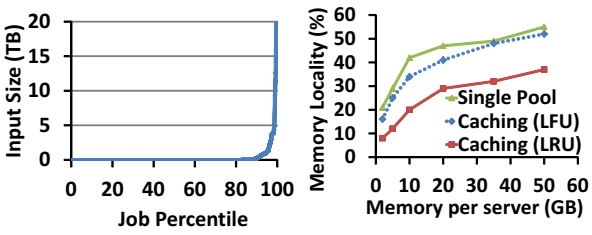
Solid state devices (SSDs) address the traditional problem of poor bandwidths of disks under random read workloads. They provide higher bandwidths and lower latency compared to disks even under random reads [5].

Despite the performance advantages of SSDs, they have not found as much traction. Recent studies show that SSDs are unlikely to be deployed as the primary medium to store the ever-growing large volumes of data in the near future [16]. The cost-per-byte of these devices have to reduce by up to three orders of magnitude for consideration. While the cost of SSDs is reducing over time (50% every year [9]), an improvement of several orders of magnitude is not predicted [8]. The deployment scenario envisioned for SSDs is to be operating in conjunction with disks [5, 7] where relatively small-capacity SSDs are used as an intermediate tier between memory and disks. Also, as we described in Section 2.2, the ever-growing demand for storage space presents further barriers to wholesale migration from disks to SSDs.

In summary, we believe that future datacenters will continue to have disks as their primary storage medium. With network speeds equalling or surpassing disk bandwidths, disk reads will be the bottleneck for tasks. A perhaps bigger implication is that the irrelevance of disk-locality again re-opens the opportunity to architect clusters, in which storage is not co-located with compute nodes [19]. Indeed, many of the earlier HPC clusters had dedicated storage servers through Storage Area Networks (SANs) and parallel file systems. Furthermore, even though disk-locality may not matter, replication schemes, such as Scarlett [18], might still provide an advantage in the cases of hot spots, as requests are load balanced to several machines.

3 Memory Locality

Based on the arguments in Section 2, we believe that focus should shift from disk-locality to *memory-locality*. As shown in Figure 1 and reported in Google's dat-



(a) Heavy-tail (b) Memory-Locality

Figure 3: **(a) Distribution of input sizes of jobs (truncated to 20TB). (b) Comparison of memory-locality between aggregated and distributed cache for different memory sizes. Despite cache hit ratios of only 52%, the heavy-tailed distribution of job input sizes leads to 64% of jobs having all tasks memory-local.**

acenters [23], reading from memory is two orders of magnitude faster than reading from disk/network. Despite the reverse deficit of an equal two orders of magnitude between the storage capacities of memory and disk, we believe memory-locality is likely to provide significant gains based on the following workload observation. In the Hadoop workload from Facebook, 96% of active jobs[2] respectively can have their entire data simultaneously fit in memory (assuming 32GB memory per server). This turns out to be due to the heavy-tailed nature of the workload, i.e., the majority of jobs access only a small fraction of the blocks (Figure 3a). Thus, very few active jobs could not fit their task data in memory.

We begin with the following strawman for memory-locality: cache each accessed block in memory (if not already present), and make way for newer blocks when out of space by evicting some of the current blocks, which are selected by the cache eviction policy. Tasks are executed memory-locally if their data is cached in memory and the corresponding machine has free computation slots. Through a trace-driven simulation of this scheme using the Hadoop logs, we highlight the challenges in achieving memory-locality. Our figure of merit is the fraction of tasks that achieve memory-locality, i.e., access their data from the memory cache, towards the goal of minimizing job completion times.

We first estimate the upper-bound for achievable memory-locality. Distribution of access counts show that 75% of the blocks get accessed only once. These single-accessed blocks, in turn, are used by 42% of the tasks. Since we cache a block in memory only after its first access, the maximum memory-locality achievable is 58%.

For our evaluation, we assume a distributed memory model where each machine maintains its own memory cache, and sixteen computation slots per machine. Figure 3b shows that, with an LFU replacement of cache blocks, 52% of tasks get memory-locality (90% of the upper-bound of 58%). The figure also compares the LRU and LFU cache replacement schemes. We see that LFU replacement is consistently better than LRU by over 15% in providing memory-locality. This is explained by big skew in access counts where 75% of the blocks are accessed only once. A block with a low access count is more unlikely to be accessed again as opposed to the oldest accessed block. As a side note, Figure 3b also includes results for a common pool model, in which all the memory is considered to be in a single machine with aggregated slots. This model barely improves upon the more realistic distributed model, indicating that the load is spread out over blocks and machines.

Finally, we proceed to see how memory-locality affects job completion times. We conservatively assume that jobs are sped-up only when all their tasks read data from memory. This is because memory-locality for only some of the tasks in the job could result in the rest becoming outliers, holding up overall completion [14]. Our evaluation shows 64% of jobs have all their tasks memory-local. This measure is conservative, but shows at the very least that those jobs would experience a reduction in job completion time. While we believe this to be an encouraging early result, there is also a large potential for further improvement, by developing new cache replacement and pre-fetching schemes.

We conclude with the following three challenges:

1. Cache Eviction Current eviction schemes are designed to improve the cache-hit ratio. But as earlier mentioned, an improvement in a job's hit ratio might not improve its completion time. We thus solicit eviction schemes that focus on reducing job completion times. Our present evaluation with the LFU replacement scheme improves 64% of the jobs out of the potential 96%. Hence, there is room for designing cache replacement schemes more tailored to the datacenter setting. Towards our goal of reducing completion times, replacement schemes should aim to increase the number of "whole" files that are present in cache.

2. Pre-fetching Blocks Tasks that access singly-accessed blocks cannot obtain memory-locality. These 42% of the tasks are distributed across 11% of the jobs. Speeding up these jobs requires an "out-of-band" mechanism that predictively pre-fetches such blocks in to memory. One option could be to load recently created data, like the output of jobs, in to memory. For jobs with multiple *waves*, we can use the first wave as a cue to load all the other input blocks to memory.

3. Scalable Cache Management File system masters that store metadata about blocks already operate under pressure [28]. An in-memory cache is expected to have

[2]By active jobs we mean jobs that have at least one task running.

higher churn in the location of blocks.Therefore, a sufficiently scalable architecture is required for caching and serving data blocks.

4 Related Work

Disk-Locality While memory-locality presents an attractive opportunity to speed-up jobs, we believe that the principles from prior work that dealt with disk-locality are still applicable. Dealing with slot contention [18] – evictions [21] versus ceding locality [26] – is pertinent when there are no free slots on a machine for a task that has its input data cached on it. The principle of replicating popular blocks [18] across memories of different machines will avoid such contentions. Even when blocks are not in memory, multiple replicas reduce contention for disk bandwidths during concurrent block accesses.

Global Memory Prior work has noted falling network latencies and the advantage of remote memory reads. The NOW project [11] proposed using a remote machine's memory as opposed to local disk. GMS [10] incorporated global memory management in a production system with resilience under churn. As network speeds are not likely to compare against local memory reads [9], we solicit a distributed cache management system as opposed to an aggregated pool. In other words, reading from a remote node's memory is unlikely to provide large gains over reading from disk due to sequential reads.

Memory Storage Systems like MMDB [20] and RAMCloud [22] envisions storing all data in memory. However, the two orders of magnitude more disk than memory capacity in many a typical data centers implies that we use memory as only a cache.

Iterative Jobs New cluster computing frameworks (Piccolo [29], and Spark [27]) are optimized to support machine learning applications and interactive data analysis, that iteratively read the same data. While the first iteration reads data from disk, these frameworks cache the data and make the subsequent iterations read from memory. Our proposal is much broader, targeting all the jobs in the datacenter (iterative as well as non-iterative) which raises a different set of challenges. Furthermore, unlike these other approaches, the in-memory cache we advocate will enable accessing data cached by other jobs.

5 Acknowledgements

We thank Dhruba Borthakur of Facebook for providing the Hadoop logs. Matei Zaharia's and Mosharaf Chowdhury's feedback helped in the draft's improvement.

References

[1] 40 and 100 Gigabit Ethernet. http://www.extremenetworks.com.

[2] An Updated Forecast of Worldwide Information Growth Through 2011. http://bit.ly/U1n6k.

[3] Applications and Organizations using Hadoop. http://wiki.apache.org/hadoop/PoweredBy.

[4] DETER. http://deter.cs.berkeley.edu/.

[5] Flash on Compute Servers. http://www.hpts.ws/session1/kleiman.pdf. *HPTPS*, 2009.

[6] Hadoop. http://hadoop.apache.org.

[7] Implications of Storage Class Memories (SCM) on Software Architectures. http://www.hpts.ws/session2/mohan.pdf. *HPTPS*, 2009.

[8] S. R. Hetzler. The Storage Chasm: Implications for the future of HDD and Solid State Storage. http://www.idema.org/.

[9] Technologies for Data-Intensive Computing. http://www.hpts.ws/agenda.html. *HPTPS*, 2009.

[10] The Global Memory System (GMS) Project. http://www.cs.washington.edu/homes/levy/gms/.

[11] The NOW Project. http://now.cs.berkeley.edu/.

[12] Dileep Bhandarkar on Datacenter Energy Efficiency, 2011. http://bit.ly/e3LVu8.

[13] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. S. Sarma, R. Murthy, H. Liu. Data warehousing and analytics infrastructure at facebook. In *ACM SIGMOD*, 2010.

[14] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, E. Harris, and B. Saha. Reining in the Outliers in Map-Reduce Clusters using Mantri. In *USENIX OSDI*, 2010.

[15] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, I. Stoica. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *USENIX NSDI*, 2011.

[16] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, A. Rowstron. Migrating Enterprise Storage to SSDs: Analysis of Tradeoffs. In *ACM EuroSys*, 2009.

[17] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *USENIX OSDI*, 2004.

[18] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, E. Harris. Scarlett: Coping with Skewed Popularity Content in MapReduce Clusters. In *ACM EuroSys*, 2011.

[19] G. Porter. Towards Decoupling Storage and Computation in Hadoop with SuperDataNodes. In *ACM LADIS*, 2009.

[20] H. Garcia-Molina and K. Salem. Main Memory Database Systems: An Overview. In *IEEE TKDE*, 1992.

[21] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair Scheduling for Distributed Computing Clusters. In *ACM SOSP*, 2009.

[22] J. Ousterhout *et al.* The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM. In *SIGOPS Operating Systems Review*, 2009.

[23] L. A. Barroso and U. Holzle. The Datacenter as a Computer. http://bit.ly/2IJd8j.

[24] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *ACM SIGCOMM*, 2008.

[25] M. Isard, M. Budiu, Y. Yu, A. Birrell and D. Fetterly. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *ACM Eurosys*, 2007.

[26] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *ACM Eurosys*, 2010.

[27] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica. Spark: Cluster Computing with Working Sets. In *USENIX HotCloud*, 2010.

[28] Kirk McKusick and Sean Quinlan. GFS: Evolution on Fast-Forward. *CACM*, March 2010.

[29] R. Power and J. Li. Piccolo: Building Fast, Distributed Programs with Partitioned Tables. In *USENIX OSDI*, 2010.