Efficiency Matters!

Eric Anderson and Joseph Tucek
Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94304, USA

ABSTRACT

Current data intensive scalable computing (DISC) systems, although scalable, achieve embarrassingly low rates of processing per node. We feel that current DISC systems have repeated a mistake of old high-performance systems: focusing on scalability without considering efficiency. This poor efficiency comes with issues in reliability, energy, and cost. As the gap between theoretical performance and what is actually achieved has become glaringly large, we feel there is a pressing need to rethink the design of future data intensive computing and carefully consider the direction of future research.

1. INTRODUCTION

Canonically instantiated by MapReduce [9], and defined in [7], data intensive scalable computing[1] (DISC) focuses on large amounts of data. Typically, DISC systems focus on performing small amounts of computing on a large amount of data rather than performing a large amount of computation on a small amount of data. DISC systems scale to thousands of nodes, analyzing many terabytes of data in a single run. They are typically built out of commodity components, and are a hot topic both of research and of commercial development.

Unfortunately, large systems come at a cost. At such scales, the probability of a component failing during a run is large, power and cooling become prohibitive, and the mere cost of acquiring such a system is daunting. We feel that these problems occur because DISC systems have repeated a mistake of old high-performance systems [1, 4]: focusing on scalability without considering efficiency. Considering a typical data intensive task, sorting, DISC systems are the performance leaders [8, 17], yet achieve only 5 to 10% of the per-node performance of non-DISC systems. Although they scale, DISC systems make poor use of individual nodes, requiring ever larger clusters to improve performance.

[1]Alternately, data intensive super computing

The gap in theoretical per node performance and what is actually achieved has become glaringly large. Expending further research effort to support such clusters (e.g., advance energy savings, analyze their workloads, etc.) is less useful if orders of magnitude of efficiency increases on the single node are not only possible, but easy. Further, inexpensive 12-, 16-, and 24-core dual-socket machines are around the corner, and will exacerbate the wastefulness of current DISC designs. We define a variety of types of efficiency, from CPU and IO to cost and human resources, and show that existing DISC systems do not perform well according to many of these measures.

We feel there is a pressing need to rethink the design of future data intensive computers and carefully consider the direction of future research because the poor efficiency of existing systems is creating avoidable problems and hiding challenges researchers should be addressing. Further, we recommend that researchers should evaluate the efficiency of the scalable systems they build, such that we get effective systems and DISC can remain data intensive rather than dismally inefficient.

2. FORMS OF EFFICIENCY

2.1 Compute Efficiency

Compute efficiency is the ability of a system to convert CPU time into useful results. For example, on a large list of randomly ordered numbers, quicksort is more compute efficient than insertion sort. As a DISC example, we improved the compute efficiency of an NFS trace analysis [2] 25× by changing the storage format and analysis code into DataSeries [2]. Since $> 95\%$ of the time in this system is spent decompressing the data, there is limited room for additional improvement.

This improvement was not unexpected, since the original analysis was written in Perl, and was converted to C++. Marceau [16] evaluated the performance and code size of a variety of languages, and determined that in general scripting languages result in significantly smaller code size at the cost of significantly higher run-times.

Another aspect of compute efficiency is not letting cycles go to waste. Presuming the existence of batch jobs for some level of fill in, a 95th precentile CPU utilization of 63% utilization [6] implies that one in three available CPU cycles is wasted; somewhat excessive even for peak provisioning. As

Year	Name	Type	Nodes	Cores	Disks	MB/s	MB/s /node	MB/s /core	MB/s /disk	bytes/s/$
2009	Hadoop	DISC	3658	29264	14632	16032.	4.45	.557	4.45	$1 \times 10^{3.2}$
2009	DEMSort	HPC	195	1560	780	9856.0	50.5	6.31	50.5	$1 \times 10^{4.1}$
2009	psort	1-node	1	1	4	96.961	96.9	96.9	96.9	$1 \times 10^{5.4}$
2008	Google (unofficial)	DISC	4000	16000?	48000	49435.	12.3	3.08	4.11	$1 \times 10^{3.5}$
2008	Hadoop *	DISC	910	7280	3640	5017.1	5.51	.689	2.75	$1 \times 10^{3.3}$
2007	CoolSort	1-node	1	2	13	108.27	108.	54.1	33.3	$1 \times 10^{5.2}$

Table 1: Selected sort benchmark results over time (sorted with newest results first). * indicates memory was sufficient for a 1-pass sort. All networks were 1 Gigabit Ethernet (GbE) except DEMSort which used InfiniBand. Google results from [8], others from [17]. Dollar values are estimates and accurate only for order of magnitude comparisons.

cores per die increase, overprovisioning of CPU will tend to worsen.

2.2 Storage Efficiency

Storage efficiency is the size in bytes of the user's data stored in a natural non-naïve format versus the raw storage actually consumed. Traditional DISC systems can perform either well or poorly by this metric. They can do poorly because they usually replicate data three times [11], achieving at most 33% space efficiency. However, they often also use compression, gaining back $3-10\times$, resulting in 100-330% efficiency. This can unfortunately reduce compute efficiency, since cycles will be spent (de)compressing the data.

However, compared to traditional redundancy techniques, this is still quite inefficient. 4-disk RAID-5 achieves 75% efficiency, and 12-disk RAID-6 arrays hit 83%. Combined with compression, this gets 250-830% efficiency. Triple replication is also used to handle node failure, but this could be more efficiently handled using mirrored RAID, cross-node ECC, or dual-homed disk arrays.

2.3 I/O Efficiency

I/O efficiency is the goodput achieved through an I/O device, such as a disk or network, compared to the theoretical maximum for the device. DISC systems often suffer from poor goodput as a result of triple replication reducing effective disk write throughput by 1/3, and the cross node replication resulting in two bytes of network traffic for each written byte. DISC network designs using commodity Ethernet usually oversubscribe by between 2:1 and 5:1, e.g., 2x10GbE and 8x1GbE uplinks per 40 1GbE nodes. Core switches are often 2:1 or more oversubscribed internally for their line cards. Oversubscription in a component is when bandwidth toward the core of the network/backplane is less than bandwidth at the edge. Finally, inefficient serialization libraries [21] can bottleneck I/O as well.

To examine the achieved I/O efficiency of DISC systems, we look at the reported results for the sort benchmark [3] since on DISC systems it exercises both disk and network I/O, and sort is an integral component of DISC frameworks. Table 1 summarizes some current sort benchmark results [17].

DISC systems extract roughly 4.5 MB/s/disk of goodput[2]. The psort system measured 100 MB/s for their drives, and DEMSort measured their drives at 80 MB/s, so we can estimate the disk I/O efficiency for DISC of about 6%. In comparison, CoolSort used laptop drives, but got 33 MB/s/disk. They measured filesystem bandwidth of 45 MB/s/disk, so achieved about 73% efficiency.

The per-node sort goodput is under 12 MB/s for all DISC systems, or 25 MB/s/node of network exchange[3]. This allows us to estimate that the 2008 Google network was 5:1 oversubscribed. Other DISC systems are at least $2\times$ less efficient in their network I/O. In comparison, DEMSort achieves 100 MB/s of exchange bandwidth, which doesn't challenge their 2 GB/s/node full-crossbar InfiniBand network, but is near the limit of 1 GbE.

2.4 Memory Efficiency

Memory efficiency is the number of bytes needed to represent the data structures and intermediate data needed for some computation compared to the best efficiency possible. For example, in Perl, representing an array of random 32 bit integers takes about 104 bytes/integer for an efficiency of $0.038\times$ relative to the minimal storage of 4 bytes. Even in C, the overhead imposed by malloc, internal fragmentation from allocation and deallocation, and the allocator's per-thread caching will result in lower memory efficiency.

Memory efficiency is important. An analysis [13] of a Microsoft DISC cluster found that 90% of short queries ($<$ 1 day) succeeded while, only 6% of lone queries ($>$ 1 year) did, attributing this mostly to resource exhaustion and contention. Given that CPU and network can't be exhausted, and disk was plentiful we speculate that they exhausted memory. Sawzall [19] uses approximate quantiles to greatly reduce and bound the memory required to determine order statistics like percentiles.

2.5 Programmer efficiency

Programmer efficiency is difficult to measure. Comparing by lines of code to solve various problems [16] shows a wide variety across languages. Code size may not be a good estimate of programmer efficiency. APL may be able to implement finding prime numbers in 17 characters, but good luck figuring out those characters. Similar problems occur in writing

[2]Sorting at 1 MB/s in a 2-pass sort requires 4 MB/s of disk I/O for the reads and writes in the two passes

[3]2-pass sort only uses the network during the first half of the computation because after the exchange nodes can operate independently

ML-style programs: the programs are shorter, but getting them to compile takes longer.

Prechelt [20] took 80 implementations of the same problem to examine programmer efficiency. In general, he found that implementation in scripting languages took about half the time. He found that the lines of code/minute for the scripting languages and system languages were comparable, and that the speedup came from halving the lines of code (e.g., through language built-ins like hash tables). Since all of the languages compared are Algol-like, this implies little about languages like APL and ML. Sawzall programs [19] tend to be $10-20\times$ smaller than the equivalent MapReduce programs.

We expect that some of the optimizations necessary to achieve better resource efficiency will come at the cost of programmer efficiency, because programmers will have to spend time thinking about how to make their algorithms and implementations more efficient. For highly reused components (e.g., the underlying DISC framework) the one time thinking cost to improve everything everywhere is worthwhile. For recurring tasks (e.g., a nightly batch job) the one time cost per site is probably worthwhile. For one-off queries, thinking time probably dominates. It may be possible that code-efficient languages like SQL or Sawzall could be automatically translated into compute and storage efficient processing systems like DataSeries [2]. Automatic optimization also tends to do better than a naïve or rushed programmer.

Finally, programmer efficiency may be negatively impacted by inefficient existing data formats or tools. Although it may be possible to rewrite those tools so that new programs can be more efficient, it clearly will increase the total programmer time.

2.6 Management Efficiency

Management efficiency is the number of people needed to run the computer infrastructure relative to the minimum required. There are few studies of this metric, because the results are so site-specific. E.g., requiring multiple administrators for 1 huge shared memory database system may be comparable to thousands of low-end commodity servers being managed by a lone administrator. However, it is generally accepted that machine/administrator ratios decrease as the machines and applications become more heterogeneous. Properly managed DISC systems can have 13 times fewer administrators per machine compared to enterprise systems [12].

We observe that increasing per-node efficiency through specialization will decrease homogeneity, as will using rarer technologies (e.g. InfiniBand). This will decrease management efficiency. Similarly, tuning a system to run close to the theoretical maximum may make it more fragile and hence increase management overhead.

2.7 Energy Efficiency

Energy efficiency is the amount of energy consumed to complete a set amount of work, including the energy to cool the system, relative to the minimal required. Since a minimum is difficult to calculate, we consider increases in efficiency relative to alternative implementations.

The power usage effectiveness (PUE) metric measures how efficiently the energy is delivered to the computer system including power distribution and cooling as overhead. PUE overhead can account for half of the TCO of a data center [5]. However, this does not include the efficiency of the work. Typical data centers are quite underused; Google, for instance, reports a mean and median CPU utilization of 30% [6]. Many components consume constant power regardless of utilization [10], resulting in a call for "energy-proportional computing" [6].

Energy efficiency is related to the other metrics, as it is increased by being more compute efficient (e.g., by turning on optimization [15]), or by reducing the amount of I/O done by reducing the replication. For example, if we assume an 8-core 24-disk system with an InfiniBand interconnect could sort 400 MB/s/node (a modest 50 MB/s/core and 66 MB/s/disk) then a cluster $90\times$ smaller would run as fast as the Hadoop cluster. Yet even if each node used $4\times$ the power of the smaller 8-core 4-disk nodes used for Hadoop, it would still be $22\times$ more energy efficient.

2.8 Cost efficiency

Cost efficiency is closely related to all of the previous efficiencies. For example, the $90\times$ reduction in nodes in Section 2.7 gives $22\times$ less capital equipment and up to $22\times$ less power/cooling infrastructure (assuming $4\times$ more expensive/power hungry nodes). Storing more bytes in less raw capacity directly reduces the cost of storage. Less standard equipment may be more cost-efficient for the capital budget. For example, InfiniBand adapters & switch ports currently cost less and provide more performance than 10GbE adapters & switch ports at the cost of a new network technology to manage. Similarly, if an application requires low-latency, non-partitionable access to a large hash table of data, it can be more cost-efficient to buy one large shared-memory machine than access memory over the network.

3. IMPLICATIONS

3.1 Reporting on efficiency

To understand the efficiency of a system, authors need to report additional data about their measurements. First, they need to report theoretical maximum performance. For example, the maximum disk-read bandwidth achievable for sequential scans, or the random I/O latency for index probes, including the benefit deep queues. For the CPU, they should estimate the maximum rate achievable, e.g., if the core data aggregation is updating an approximate quantile in a cube, and that piece has been well optimized, then what the processing rate would be if only that operation was performed. Similar data for the other forms of efficiency are useful if they are relevant to the algorithm choice, e.g., data sizes close to the transition between one- and two-pass sorts.

Further, "for better or for worse, benchmarks shape a field". In this paper we were forced to compare DISC systems mostly based on sort, because that was the only benchmark in common between all of them. This narrow a set of benchmarks provides a poor basis for good comparisons. Therefore we believe that the field needs a broader set of benchmarks, akin to SPEC, TPC, or SPLASH-II.

If the application runs across multiple nodes, it is important to understand the effect of bisection bandwidth limits. This limitation can be estimated by first scaling the program across cores, and then across nodes, since cross-core bisection bandwidth is about 10× higher than specialized networks and about 100× higher than commodity networks.

3.2 Distorting research

The lack of efficiency in existing DISC systems is distorting the research that is being done, both by creating problems that would not exist in more efficient systems and by hiding problems that should exist. For example, low efficiency leads to huge systems with long-running jobs. Hence, the MTTF of the underlying commodity components may be exceeded. The expectation of failures results in additional redundancy being added, further reducing efficiency.

Similarly, the high power requirements from the large systems result in a desire to run the systems hot, which reduces reliability. Further, this increases the desire for power-proportionality since many components are under-utilized. Finally it causes researchers to examine using laptop or embedded processors, because the applications are unable to run server processors at full rates, sacrificing the potential to reduce component counts.

The inefficient network utilization creates a need to emulate 4,000 node crossbar switches through complex fat-tree topologies, and results in excess focus exploiting rack-level locality. Increasing efficiency by 20× would allow the use of inexpensive (< $800/node) 2 GB/s/node full crossbar switches that are limited to about 300 ports.

Finally, the inefficient systems create an excessively low bar for comparisons, allowing researchers to claim success when they achieve a few MB/s in a new system as this is comparable to existing ones.

More efficient DISC systems will create new research directions. We describe a few possibilities here. First, there will be research in creating more efficient systems. The greater than 10× possible improvement in component count results in a much smaller system, which could then benefit from more reliable components. The CPU/memory/disk/network balance in such systems is also likely to change.

Second, as different tasks have different resource draws (memory, CPU, disk), there are opportunities to increase efficiency through heterogeneous clusters. This would present new management issues, as well as opportunities to migrate tasks to appropriate hardware.

Third, if the per-node analysis rate is high (1 GB/s/node), then an application that requires an exchange will need over ten 1GbE nodes to have the exchange bandwidth to match the performance of one node. Compressing data prior to transmission may become valuable. The trend of increasing core count is likely to exacerbate this challenge, and the challenge of getting disk bandwidth given the physical constraints on disks in a 2U server.

4. RELATED WORK

Joseph Hellerstein observed [14] that Hadoop and Greenplum are 40× apart in node counts. He primarily ascribes this to node type choices (2-socket & 4-disk vs. 4-socket & 48-disk), and secondarily to Hadoop's choice to write data redundantly rather than recompute. Without more details, we can't tell whether either is doing well, but it seems likely both are inefficient. Indeed, one comment to the posting notes Fox Interactive Media loading 4TB/hour into a 40 node Greenplum system as being "unmatched performance." However, this utilization is only 28 MB/s/node, which shows a <30% network utilization, even if the nodes only have 1 GbE port.

Pavlo, et al. [18] compared Hadoop with two parallel databases, Vertica and DBMS-x. They found that the load times were much better for Hadoop, while the query times for SQL-like queries were better on the parallel databases (especially for indexed queries). The results for user-defined functions were mixed, but the authors say that it was much more difficult to implement them in the parallel databases. Without more detail, we can only poorly estimate the efficiency of the systems. In the load experiments, the best case Hadoop load rate is 30 MB/s and the normal case is 7 MB/s, or 20% and 5% efficiency assuming that the disk write rate is the same as the read rate. The parallel databases are at least 4× worse. For grep, the efficiency of the databases is better: they achieve 54 MB/s, or at most 36% efficiency, since they compress the data and achieve an unknown compression improvement.

Bowen Alpern, et al. [1] observed some of these issues applying to parallel systems back in 1995. In particular, they observed that parallel efficiency drops if node count is increased but problem size is held constant. He also observed the pernicious effect that putting the linear scalability line on paper graphs results in people artificially scaling problem size and devoting more programming effort to the larger processor counts.

5. CONCLUSIONS

We have examined many different forms of efficiency that are present in DISC systems, and have shown that most existing systems do not perform well according to these measures. This poor efficiency is distorting the research that is being done, both by creating problems that need not exist, and by hiding problems that should exist. Therefore we have recommended that researchers evaluate the efficiency of the systems that they build, such that we get effective systems and DISC can remain data intensive scalable computing rather than dismally inefficient scalable computing.

6. ACKNOWLEDGEMENTS

We would like to thank Mehul Shah, Mark Lillibridge, Kim Keeton, our shepherd Dushyanth Narayanan, and the anonymous reviewers for their feedback, which improved the quality of our arguments.

7. REFERENCES

[1] Bowen Alpern and Larry Carter. The myth of scalable high performance. In *PPSC 1995: SIAM Conference on Parallel Processing for Scientific Computing*, pages

857–859, February 1995. Available at http://www.cs.ucsd.edu/users/carter/Papers/scale.ps Accessed June 2009.

[2] Eric Anderson, Martin Arlitt, Charles B. Morrey III, and Alistair Veitch. DataSeries: An Efficient, Flexible Data Format for Structured Serial Data. *Operating Systems Review*, 43(1):70–75, 2009.

[3] Anon, et. al. A measure of transaction processing power. *Datamation*, 31(7):112–118, 1985. Available at http://sortbenchmark.org/AMeasureOfTransactionProcessingPower.doc accessed June 2009.

[4] David H. Bailey. Highly parallel perspective: Twelve ways to fool the masses when giving performance results on parallel computers. *Supercomputing Review*, 4(8):54–55, Aug 1991.

[5] Luiz André Barroso. Saving the planet with systems research. Keynote abstract at http://www.cs.virginia.edu/asplos09/keynote.htm Accessed June 2009.

[6] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[7] Randal E. Bryant. Data-intensive supercomputing: The case for DISC. Technical Report CMU-CS-07-128, Carnegie Mellon University, 2007.

[8] Grzegorz Czajkowski. Sorting 1pb with MapReduce. Available at http://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce.html Accessed June 2009.

[9] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI '04: Proceedings of the sixth Conference on Operating Systems Design and Implementation*, pages 10–10. USENIX Association, 2004.

[10] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual International Symposium on Computer Architecture*, pages 13–23, New York, NY, USA, 2007. ACM. Available at http://research.google.com/archive/power_provisioning.pdf Accessed June 2009.

[11] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *SOSP '03: Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 29–43. ACM Press, 2003.

[12] James Hamilton. Service design best practices. Available at http://www.mvdirona.com/jrh/TalksAndPapers/JamesHamilton_POA20090226.pdf Accessed September 2009.

[13] Bingsheng He, Mao Yang, Zhenyu Guo, Rishan Chen, Wei Lin, Bing Su, Hongyi Wang, and Lidong Zhou. Wave Computing in the Cloud. *HotOS 2009*, 2009. Available at http://www.usenix.org/events/hotos/tech/full_papers/he/he.pdf Accessed June 2009.

[14] Joseph M. Hellerstein. Diverging views on Big Data density, and some gimmes. Available at http://databeta.wordpress.com/2009/05/14/bigdata-node-density/ Accessed June 2009.

[15] Energy-efficient software guidelines. Available at http://software.intel.com/en-us/articles/energy-efficient-software-guidelines/ Accessed June 2009.

[16] Guillaume Marceau. The speed, size and dependability of programming languages. Available at http://gmarceau.qc.ca/blog/2009/05/speed-size-and-dependability-of.html Accessed June 2009.

[17] Chris Nyberg and Mehul Shah. Sort benchmark home page. Available at http://sortbenchmark.org Accessed June 2009.

[18] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. Dewitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *SIGMOD '09: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, June 2009. Available at http://db.csail.mit.edu/pubs/benchmarks-sigmod09.pdf Accessed June 2009.

[19] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming*, 13(4):277–298, October 2005. Available at http://labs.google.com/papers/sawzall.html accessed June 2009.

[20] Lutz Prechelt. An empirical comparison of seven programming languages. *Computer*, 33(10):23–29, 2000. Available at http://page.mi.fu-berlin.de/~prechelt/Biblio/jccpprt_computer2000.pdf accessed June 2009.

[21] Benchmarking - thrift-protobuf-compare. Available at http://code.google.com/p/thrift-protobuf-compare/wiki/Benchmarking Accessed June 2009.