Applying Idealized Lower-Bound Runtime Models to Understand Inefficiencies in Data-Intensive Computing

Elie Krevat[*], Tomer Shiran[*], Eric Anderson[†], Joseph Tucek[†], Jay J. Wylie[†], Gregory R. Ganger[*]
[*]Carnegie Mellon University [†]HP Labs

Categories and Subject Descriptors

D.4.8 [**Operating Systems**]: Performance—*Measurements, Modeling and prediction, Operational analysis*; C.4 [**Performance of Systems**]: Modeling techniques

General Terms

Measurement, Performance

1. INTRODUCTION

"Data-intensive scalable computing" (DISC) refers to a rapidly growing style of computing characterized by its reliance on large and expanding datasets [3]. Driven by the desire and capability to extract insight from such datasets, DISC is quickly emerging as a major activity of many organizations. Map-reduce style programming frameworks such as MapReduce [4] and Hadoop [1] support DISC activities by providing abstractions and frameworks to more easily scale data-parallel computations over commodity machines.

In the pursuit of scale, popular map-reduce frameworks neglect efficiency as an important metric. Anecdotal experiences indicate that they neither achieve balance nor full goodput of hardware resources, effectively wasting a large fraction of the computers over which jobs are scaled. If these inefficiencies are real, the same work could be completed at much lower costs. An ideal run would provide maximum scalability for a given computation without wasting resources. Given the widespread use and scale of DISC systems, it is important that we move closer to frameworks that are "hardware-efficient," where the framework provides sufficient parallelism to keep the bottleneck resource fully utilized and makes good use of all I/O components.

An important first step is to understand the degree, characteristics, and causes of inefficiency. We have a simple model that predicts the idealized lower-bound runtime of a map-reduce workload by assuming an even data distribution, that data is perfectly pipelined through sequential operations, and that the underlying I/O resources are utilized at their full bandwidths whenever applicable. The model's input parameters describe basic characteristics of the job (e.g., amount of input data), of the hardware (e.g., per-node disk and network throughputs), and of the framework configuration (e.g., replication factor). The output is the idealized runtime.

The goal of the model is not to accurately predict the runtime of a job on any given system, but to indicate what the runtime theoretically *should* be. To focus the evaluation on the efficiency of the programming framework, and not the entire software stack, measured values of I/O bandwidths are used as inputs to the model, or predicted values are used if the actual hardware is not available. Indeed, our analysis of a number of published benchmark results, on presumably well-tuned systems, reveal runtimes that are 3–13$\times$ longer than the ideal model suggests should be possible (due to space constraints, see the full length paper [5]).

We haven't yet determined why well-used map-reduce frameworks exhibit large slowdowns, although there are many possibilities. Instead, we focus on a limited but efficient parallel dataflow system called Parallel DataSeries (PDS), which lacks many features of other frameworks, but its careful engineering and stripped-down feature-set demonstrate that near-ideal hardware efficiency (within $\sim$20%) is possible. PDS is used to explore the fundamental sources of inefficiency common to commodity disk and network-dependent DISC frameworks; any remaining inefficiencies of popular map-reduce systems must then be specific to the framework or additional features that are not part of PDS (e.g., distributed file system integration, dynamic task distribution, or fault tolerance). Our experiments point to straggler effects from disk-to-disk variability and network slowdown effects from the all-to-all data shuffle as responsible for the bulk of PDS's inefficiencies, and therefore a performance issue for DISC frameworks in general.

2. PERFORMANCE MODEL

The idealized runtime model is intended for data-intensive workloads where computation time is negligible in comparison to I/O speeds, which applies to many grep- and sort-like jobs that are representative of MapReduce jobs at Google [4]. Other simplifying assumptions are that the network is capable of full bisection bandwidth, input data is distributed evenly across a homogeneous set of nodes, and one job runs at a time.

The model calculates the total time by breaking a map-reduce dataflow into two pipelined phases, where the pipeline is as fast as the slowest I/O component in each phase. The first phase of a parallel dataflow reads data, processes it in the map operator, and shuffles it over the network. A barrier occurs just before the local sort at the destination node, separating the two phases, since the last element to arrive may in fact be the first to go out in sorted order. The second phase begins with a local sort, passes data through the reduce operator, and writes data back to disk (possibly replicated).

Inputs to the model include the amount of input data (i), the speed of a disk read (D_r), disk write (D_w), and network transfer (N), and the amount of data flowing through the system at any time. The full model accounts for different configurations of a parallel map-reduce, including data replication levels, in-memory vs. 2-phase local sorts, and optional backup writes after the map phase. Due to space constraints, we only include a simplified version of our model that applies to a parallel sort with no additional replication, an in-memory sort, and no backup write:

$$t_{sort} = \frac{i}{n}\left(max\left\{ \frac{1}{D_r}, \frac{n-1}{nN} \right\} + \frac{1}{D_w} \right)$$

In this equation, the time to complete the first phase is just the maximum of the two pipelined operations of reading data from local disk ($\frac{1}{D_r}$) and shuffling the data over the network. The $\frac{n-1}{n}$ term appears in the equation, where n is the number of nodes, because in a perfectly-balanced system each node partitions and transfers that fraction of its mapped data over the network, keeping $\frac{1}{n}$ of the data for itself. The time for the second phase is then just the time taken to write the output data back to local disk ($\frac{1}{D_w}$). Parameter values are determined via measurements through the OS, so any inefficiencies are those of the programming framework.

3. EXPLORING EFFICIENCY OF DISC

We performed disk and network microbenchmarks to measure the maximum achievable I/O speeds on our cluster, where each node consists of two quad-core Intel Xeon processors, 16 GB of RAM, and a Seagate Barracuda ES.2 SATA drive. Microbenchmark results show a large variation of disk and network shuffle speeds across homogeneous nodes, contributing to slower PDS parallel sorts than computed with our idealized model. The straggler problem is well-known [2] and was not unexpected. However, our approach to investigating stragglers is to understand why they fundamentally exist and how much of their impact is necessary. With PDS, we can achieve within 21% of the model's idealized runtime.

Disk microbenchmarks: Bandwidth is measured with the `dd` utility and large file and block sizes. On a single node, XFS filesystems have consistent read performance (averaging 107 MB/s) and varying writes between 88–100 MB/s (averaging 99 MB/s). However, the speeds can vary over the entire cluster from 102–113 MB/s for reads and 87–107 MB/s for writes, both with a standard deviation of 2. Some of this variability in bandwidth of otherwise identical make and model disks is due to XFS placement policies, but the majority is due to a technique in modern disks that we refer to as *adaptive zoning* [6], where traditional zoning schemes have been extended to maximize platter densities post-production according to the capabilities of each disk head.

Network microbenchmarks: With our all-to-all network microbenchmark that analyzes just the network component of a map-reduce shuffle, we measured bi-directional bandwidth between two nodes of 112 MB/s over 1 Gbps Ethernet. The resulting throughput for completing a larger 4-way shuffle drops under 104 MB/s, and then slowly decreases to 98 MB/s at 32 nodes and under 94 MB/s at 64 nodes. Some of TCP's unfairness and stability issues are known and prompting further research, however, we have found no previous research specifically targetting TCP's all-to-all network performance in high-bandwidth low-latency environments. The closest other issue that we know of is the incast problem [7], a 2–3× order of magnitude collapse in network bandwidth during an all-to-one pattern of synchronized reads.

PDS sort evaluation: Parallel DataSeries is an efficient and flexible data format and library that supports passing records in a parallel and pipeline fashion through a series of modules. As presented in Figure 1 for up to 41 nodes, PDS parallel sort times increase from 8% to 21% longer than an ideal model input with measured disk and network bandwidth. On a single node, about 4% of lost efficiency is from a known data expansion when converting inputs. The in-memory sort takes about 2 seconds, which accounts for another 2–3% of the overhead. These two factors explain the majority of the 8% overhead of the single node case.

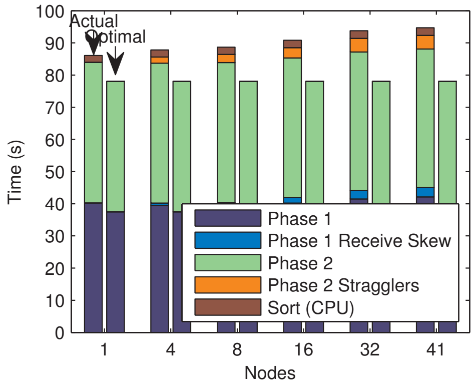

Figure 1: Using Parallel DataSeries to sort up to 164 GB over 1 Gbps Ethernet, completion times slowly increase from 8% to 21% higher than an idealized lower-bound prediction that inaccurately assumes full disk and network performance.

The growing divergence of larger parallel sorts from the model is explainable by a number of factors, including disk stragglers and network skew and slowdown effects. The *Stragglers* category in the time breakdown accounts for 5% additional overhead and represents the average time that a node wastes by waiting for the last node in the job to finish. Most of the straggler effects broken out in the figure are caused by the different disk write speeds from Phase 2 of the map-reduce, as the sort barrier forces synchronization before a node completes Phase 1. A similar disk skew occurs in Phase 1 from reading data from disk, but it is further complicated by delays in the network shuffle. Broken out in the figure is a 4% overhead from the *Phase 1 Receive Skew*, which is the average difference of each node's last incoming flow completion time and its average incoming flow completion time. This skew suggests an unfairness across network flows, and is compounded by a general slowdown in network speeds that causes a delay in the completion of Phase 1.

Building a balanced system becomes more complicated with the observed disk and network effects. It was surprising that even with a perfectly balanced workload, without any data skew, there was still a significant effect from disk stragglers. Additional experiments with a faster network showed that disk stragglers are not a continuously growing effect, but they are present, and even one particularly slow node can have a large impact. Our system configuration initially pointed to disk bandwidth as the slowest component. However, the results of our network microbenchmark are confirmed by the actual performance of PDS—the slower network shuffle speeds for Phase 1 become the bottleneck at around 16 nodes and are responsible for the growing divergence from the model.

4. REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org/.
[2] G. Ananthanarayanan, et al. Reining in the Outliers in Map-Reduce Clusters using Mantri. OSDI, 2010.
[3] R. E. Bryant. *Data-Intensive Supercomputing: The Case for DISC*. Technical report. 2007. CMU-CS-07-128.
[4] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 2008.
[5] E. Krevat, et al. *Applying Simple Performance Models to Understand Inefficiencies in Data-Intensive Computing*. Technical report. 2011. CMU-PDL-11-103.
[6] E. Krevat, et al. Disks Are Like Snowflakes: No Two Are Alike. HotOS, 2011.
[7] A. Phanishayee, et al. Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems. FAST, 2008.