Recommending Just Enough Memory for Analytics

Charles Reiss and Randy H. Katz

1 Introduction

MapReduce was designed by Google for large-scale data analysis on slow but cheap disk-based storage. Nevertheless, memory has declined in price to where cost-effective machines offer ever larger memory capacity. Furthermore, a more diverse data analyst community, with smaller datasets, has emerged. These trends motivate new parallel processing frameworks, like Spark [2], with better support for in-memory data analysis.

For these frameworks, memory capacity is a bottleneck. Once an application has "enough" memory, adding more rarely improves performance. But underallocating memory often prevents programs from completing in a reasonable time, if at all. To avoid this, users often pay for memory they cannot effectively use, and are reluctant to adjust working allocations.

We are developing tools for Spark (and similar frameworks) that let users better understand their memory requirements. They give feedback, informed by runtime monitoring, so users can adjust their configurations to lower their costs. Unlike prior work, our estimation uses a single run, and can be effective even when that run has too little memory.

2 Runtime Model

Spark's *block manager* tracks the location and size of each data partition distributed across the cluster. Spark evicts data when insufficient memory is available. Data that is reaccessed is read from disk and recomputed. The block manager's space should be sized large enough so that "hot" data will never be evicted. Spark programs also need some other memory to run the application code, but usually only a modest amount.

We instrument Spark primarily through the block manager, recording all reads and writes. We filter these records to remove recomputation activity, yielding a trace that is independent of the actual memory configuration and replacement policy. To enable filtering, we annotate accesses with the responsible task, so we can identify activity caused by an unsuccessful read.

To form our memory recommendation, we use algorithms originally developed for processor caches [1] to estimate the working set size and evaluate the effect of smaller configurations. We apply these to the block manager traces, after filtering and classifying data, combined with a model of how Spark evicts data.

We make a configuration recommendation after a single run. This includes a type and number of workers, and the portion of memory managed by the block manager. We leave aside enough unmanaged memory for transient copies of a partition per core, and a fixed amount for application code. We set aside enough managed space on each worker for broadcast data and to handle imperfect balancing of the partitions. Then, we recommend enough workers to satisfy the working set requirement with the remaining managed space, suggesting the worker type giving minimum cost per core.

3 Results

We evaluated a prototype on several Spark programs. For comparison, we recorded program runtimes over a spectrum of configurations. We examined our recommendations (for each parameter), and compared them to the "knee" of the resource/performance curve. For programs whose memory usage is dominated by Spark-managed datasets, our recommendations were slightly above that required for efficient execution.

4 Future Work

With extensions, our approach can serve a wider variety of users. Providing advice after one run is not fast enough if the program is only run once. But, to use recommendations on-the-fly, frameworks must be able to adjust configurations and move data without restarting. More fundamentally, we assume that programs operate best with a conservative memory allocation. But some programs have meaningful memory/cost tradeoffs. We estimate the impact of smaller memory sizes, but more work is needed to express these estimates in units that users would find useful for understanding their tradeoff between memory and run-time.

SoCC'13, 1–3 Oct. 2013, Santa Clara, CA, USA
ACM 978-1-4503-2428-1/13/10.
http://dx.doi.org/10.1145/2523616.2525957

References

[1] T. Kelly and D. Reeves. Optimal web cache sizing: Scalable methods for exact solutions. In *Fifth Int'l Web Caching and Content Delivery Workshop*, 2000.

[2] M. Zaharia et al. Spark: Cluster computing with working sets. In *HotCloud '10*, 2010.