Task Fusion: Improving Utilization of Multi-user Clusters

Robert Dyer
Iowa State University
rdyer@iastate.edu

Abstract

Researchers use shared computing clusters to ask interesting questions and wish to maximize their utilization. Currently, optimizations focus on individual programs. We present *task fusion* to automatically merge multiple tasks into a single task. An example implementation shows fused tasks take 14–90% less time than running the tasks individually.

Categories and Subject Descriptors D.1.3 [*PROGRAMMING TECHNIQUES*]: Concurrent Programming

Keywords MapReduce; optimization; task fusion

1. Introduction

Big data techniques allow researchers to propose and answer many interesting hypotheses on very large sets of data. Analyzing big data is accomplished using a distributed cluster, which is typically shared by several users. Increasing the utilization of such shared resources is important in order to keep costs low and allow for more users.

Many techniques have previously been proposed to optimize programs that run on such clusters, however these optimizations typically focus on single programs. Assuming we have already used existing single-program optimization techniques to increase utilization of shared clusters, important research questions we wish to answer are: Can we further increase utilization by automatically optimizing groups of programs? If so, what pre-conditions must hold to apply such cross-program optimizations?

In this research we answer these questions and propose a new optimization technique called *task fusion*, which is inspired by a compiler optimization called loop fusion that increases data locality by fusing multiple loops together. Task fusion takes multiple, individual tasks and fuses them into a single task. This process creates a single program that reads the input data only once and performs multiple computations on it, thus avoiding the overhead of reading the data multiple times.

We prove the feasibility of our optimization by implementing it in the Boa framework for software repository mining [4, 6]. Our early evaluation shows that task fusion in Boa provides performance benefits of up to ten times faster execution when compared to running tasks individually.

2. Background and Related Works

Dean and Ghemawat described a MapReduce framework [3] for distributed computations. In this framework, data is represented as key/value pairs. Each pair is given to a map (or *mapper*) function, which transforms it into zero or more output key/value pairs. This output is then fed into a second user-defined function called a *reducer*, which aggregates all values for a given key and produces a final result. The framework automatically sorts the data between the map and reduce phases. Hadoop [1] is an open-source implementation of MapReduce written in Java.

Chain folding and job merging [5] are two optimizations for MapReduce programs. *Chain folding* aims to take multiple map or reduce steps in a single program and fold them together into fewer steps. *Job merging* is when two un-related programs that operate over the same input are manually merged into a single program by merging the mappers and reducers together.

FlumeJava [2] is a library for Java that abstracts the details of MapReduce from the user, instead providing a set of high-level features such as parallel collections and functions. The library generates a query plan which may include multiple MapReduce phases. The query plan is optimized using *sibling fusion* and *MSCR fusion*, which are similar to chain folding and job merging, to reduce the total number of MapReduce tasks executed. This optimization is for a single FlumeJava program.

3. Task Fusion

The previously mentioned optimizations typically focus on individual programs. The only optimization that looks at multiple programs, job merging, assumes the programs are all by the same author and requires manually merging them together. In this section we propose a fully automated solu-

SPLASH '13, October 26–31, 2013, Indianapolis, Indiana, USA.

ACM 978-1-4503-1995-9/13/10.
http://dx.doi.org/10.1145/2508075.2514878

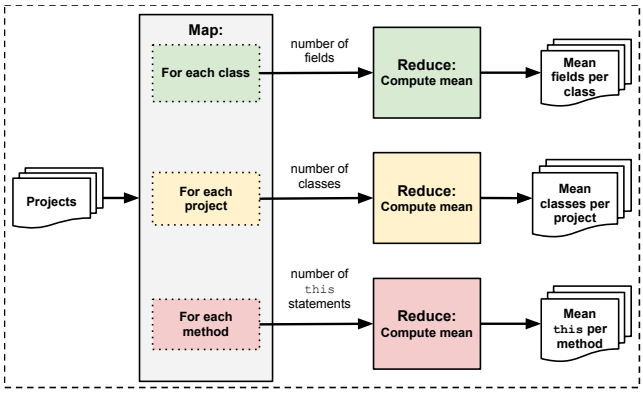

Figure 1. Three example tasks fused into a single MapReduce program, with 1 mapper and 3 reducers.

tion called *task fusion* that takes unrelated programs, possibly from different users in a shared cluster, and fuses them into a single task.

As an example, consider three MapReduce tasks to mine the mean number of fields per class, mean number of classes per project, and mean number of `this` statements per method. For these tasks, the mappers iterate over components and output values for each component. The reducers then compute the mean value across components.

With task fusion, the three separate MapReduce tasks are fused into a single MapReduce task (Figure 1). The map function is a composition of the three individual task's map functions. There are also multiple reducers, one per original task. One problem is ensuring the values output from the map task are routed to the correct reducer, thus giving the same (separate) output files as when running the tasks individually.

To solve this problem, we rewrite the map task's output calls to use a composite key and provide a custom partitioner. Partitioners take map output and routes it (based on the key) to reducers. Our partitioner uses the composite key to ensure the output from a map task goes to the correct reducer, ensuring the same final three outputs.

Currently task fusion only works if all tasks share the same input data, have no dependency conflicts, use no shared state, and do not have side effects (such as writing to files). Relaxing these assumptions to allow application to a wider range of systems is currently future work.

4. Evaluation

We evaluate the effectiveness of task fusion by implementing it in the Boa infrastructure [4, 6] for software repository mining. Boa programs compile to Hadoop MapReduce programs and we modified the compiler to use task fusion when more than one program is given as input.

We executed over 60 Boa tasks from several previous studies. These tasks have varying complexity and their execution takes anywhere from 25 seconds up to 24 minutes each. We created workloads containing 21 fast tasks, 22 medium sized tasks, 18 slow tasks, and a mixed workload of 9 tasks containing an equal number of fast, medium, and slow tasks. The results of running these workloads are shown in Table 1.

	Sequential	Task Fusion	Speedup
Fast	486	45	10.80X
Medium	8,377	6,601	1.27X
Slow	16,642	14,211	1.17X
Mixed	4,687	3,255	1.44X

Table 1. Execution times (in seconds) for workloads.

The first column shows the time (in seconds) to run the tasks in a given workload individually. The next column shows the execution time when the tasks are fused together. The last column shows the speedup achieved by task fusion.

In the best case, task fusion is over ten times faster than running the tasks individually. Even in the worst case, task fusion runs in 14% less time than individually. These early results clearly demonstrate the potential benefit of using task fusion in shared clusters.

5. Conclusions and Future Work

Maximizing the utilization of shared clusters is important for researchers interested in answering questions with big data techniques. Task fusion is an optimization that allows multiple programs from different users to be fused and executed together, requiring 14%–90% less time than executing the tasks individually.

In the future we plan to investigate ways to relax some of the assumptions currently required for task fusion, allowing application of the technique to more infrastructures.

Acknowledgments

This work funded in part by NSF grants CCF-10-17334 and CCF-11-17937.

References

[1] Apache Software Foundation. Hadoop: Open source MapReduce. `http://hadoop.apache.org/`.

[2] C. Chambers, A. Raniwala, F. Perry, S. Adams, R. R. Henry, R. Bradshaw, and N. Weizenbaum. FlumeJava: easy, efficient data-parallel pipelines. In *PLDI'10*, pages 363–375, 2010.

[3] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *OSDI'04*, 2004.

[4] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *ICSE'13*, pages 422–431, 2013.

[5] D. Miner and A. Shook. *MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems*. O'Reilly Media, 2012.

[6] H. Rajan, T. N. Nguyen, R. Dyer, and H. A. Nguyen. Boa website. `http://boa.cs.iastate.edu/`, 2012.