Department of Computer Science

University of Reading

Implementing the MapReduce architecture for small datasets

Dan Hughes

Background

The MapReduce paradigm is a distributed data-processing framework for processing large amounts of data using a two-step execution of parallel functions. Designed at Google for processing log files [1], it enables commodity hardware to be added to a resource pool of machines.

The open-source implementation of MapReduce is a collection of software called Hadoop. This consists of a number of individual applications for resource management, file storage and other tasks. As a results, it has a significant set-up process and resource overheads.

This project presents an alternative platform to Hadoop, to be built entirely in JavaScript using the NodeJS runtime.

Objectives

- The project should be a rapidly deployable platform for prototyping MapReduce algorithms on commodity hardware.
- Each individual node should automatically tune its own performance parameters, ensuring that slow hardware is able to contribute.
- Data should be processed with streaming operators, to ensure that memory limits on low-power devices are not exceeded.

Algorithm

The general MapReduce algorithm follows this structure:

- Each node is assigned a subset of the data.
- Each node applies a map function in parallel to the data to build a list of key-value pairs.
- Each node is assigned all the values for a given key.
- Each node applies a reduce function in parallel to convert all the values for a given key to a single result.
- The results are output to file.

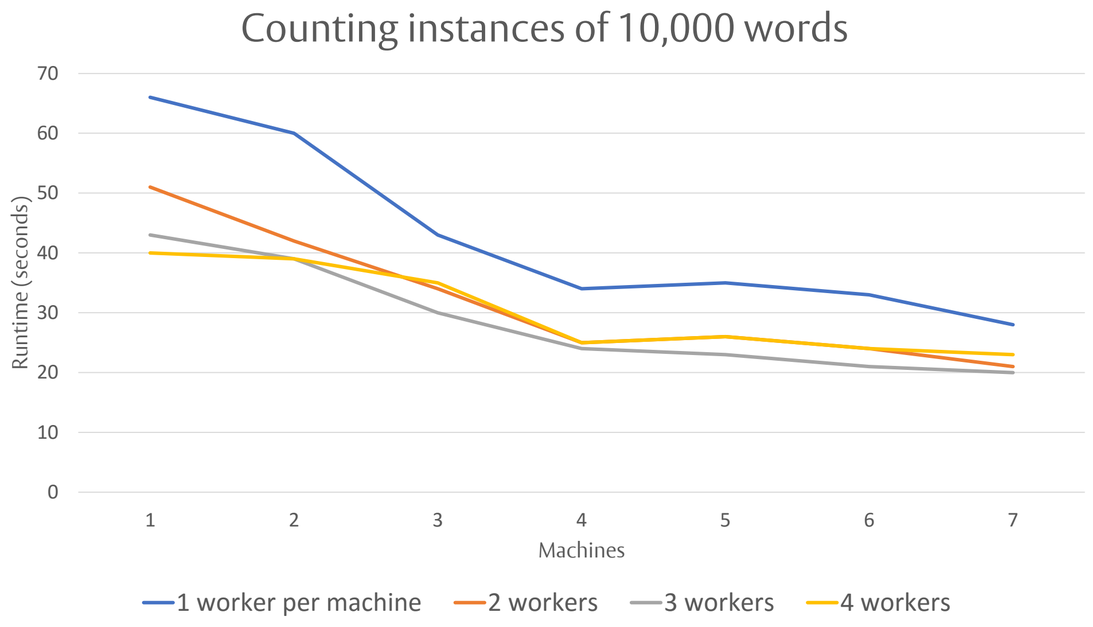
Implementation

All nodes connect to the controller over a TCP WebSocket [2] to allow for real-time streaming of data. The controller generates a queue of work tasks to be processed by streaming the input file through a user-supplied stream transformation function. This allows the controller to always have work to be processed without overloading the machine's memory.

The worker nodes apply a map function to every value, store the value results in local memory, and send back the resulting key to the controller. Through this mechanism, the controller builds a list of which nodes have values for each key. The nodes also apply a user-supplied combination function, to perform local reduction of values. This allows for the network as a whole to act as a distributed object store, accessible to every node within it.

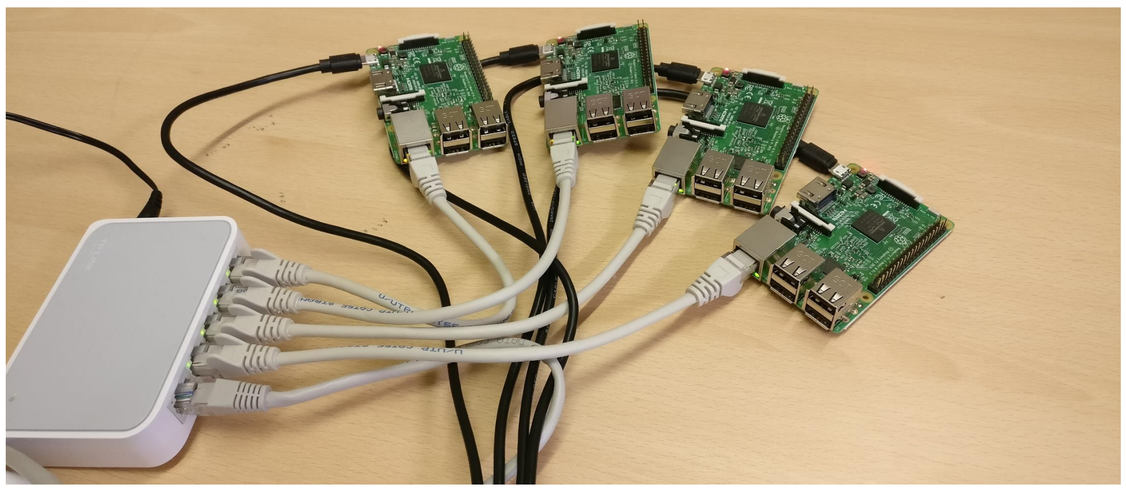
The controller then allocates each key to a node. Each node queries the distributed object store to find all the values for its assigned keys, applies the a reduction function, and returns these value to the controller.

Scalability



Applications

The purpose of the system is to allow users to prototype MapReduce algorithms on any available commodity hardware. It has been tested on systems ranging from a cheap Raspberry Pi cluster to powerful desktop systems. It has been applied to distributed classification problems such as K-Nearest Neighbour, and simpler algorithms such as calculating mutual friends for each pair of users in a large social graph.



References

1. J. Dean and S. Ghemawat, 'MapReduce: Simplified data processing on large clusters', OSDI 2004
2. [RFC6445] I. Fette, "The WebSocket Protocol", RFC6445, December 2011

Degree programme

BSc Computer Science with Industrial Year