How to Implement MapReduce Using Gearman

Presented By Jamie Pitts

Thursday, December 9, 2010
A Problem Seeking A Solution

Given a corpus of html-stripped financial filings: Identify and count unique subjects.

Possible Solutions:

1. Use the unix toolbox.
2. Create a set of unwieldy perl scripts.
3. Solve it with MapReduce, perl, and Gearman!
What is MapReduce?

A programming model for processing and generating large datasets.

The original 2004 paper by Dean and Ghemawat

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to compute day in day out. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the many details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the map and reduce primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a map operation to each logical “record” in our input in order to
M.R. Execution Model

A large set of input key/value pairs is reduced into a smaller set of output key/value pairs

1. A Master process splits a corpus of documents into sets. Each is assigned to a separate Mapper worker.

2. A Mapper worker iterates over the contents of the split. A Map function processes each. Keys are extracted and key/value pairs are stored as intermediate data.

3. When the mapping for a split is complete, the Master process assigns a Reducer worker to process the intermediate data for that split.

4. A Reducer worker iterates over the intermediate data. A Reduce function accepts a key and a set of values and returns a value. Unique key/value pairs are stored as output data.
M.R. Execution Model Diagram

Figure 1: Execution overview
Applying MapReduce
Counting Unique Subjects in a Corpus of Financial Filings

Corpus of Filings, Split by Company ID

Intermediate Data: Redundant records of Subjects and Counts

Reducer

Output Data: Unique Records of Subjects and Counts

Mapper
What is Gearman?

A protocol and server for distributing work across multiple servers.

- Job Server implemented in pure perl and in C
- Client and Workers can be implemented in any language
- Not tied to any particular processing model
- No single point of failure

Current Maintainers: Brian Aker and Eric Day
Gearman Architecture

- Job Server is the intermediary between Clients and Workers.
- Clients organize the work.
- Workers define and register work functions with Job Servers.
- Servers are aware of all available Workers.
- Clients and Workers are aware of all available Servers.
- Multiple Servers can be run on one machine.
Perl and Gearman

Use Gearman::XS to Create a Pipeline Process

http://search.cpan.org/~dschoen/Gearman-XS/
Perl and Gearman
Client-Server-Worker Architecture

Gearman Job Servers

```
bash-3.2s ./sbin/gearmand -v --port=4730 --log-file=logs/4730.log &
[1] 81764
bash-3.2s ./sbin/gearmand -v --port=4731 --log-file=logs/4731.log &
[2] 81766
```

Each computer has two Gearman Job Servers running:
one each for handing mapping and reducing jobs to available workers.

Gearman::XS::Client

Split > Do Mapping > Do Reducing

The Master uses Gearman clients to control the MapReduce pipeline.

Gearman::XS::Worker

Mappers
Reducers

Mapper and Reducer workers are perl processes, ready to accept jobs.
Example
Processing Filings from Two Companies

Mapper
Reducer

Filings from Two Companies

Intermediate Data:
Redundant records of Subjects and Counts

Output Data:
Unique Records of Subjects and Counts

Thursday, December 9, 2010
Example: The Master Process
Setting Up the Mapper and Reducer Gearman Clients

use Gearman::XS qw(:constants);
use Gearman::XS::Client;
use FreezeThaw qw(freeze thaw cmpStr safeFreeze cmpStrHard);

# set up the mapper
my $mapper = new Gearman::XS::Client;
if ($mapper->add_server('localhost', 4730) != GEARMAN_SUCCESS) {
    printf(STDERR "%s\n", $mapper->error());
    exit(1);
}

# set up the reducer, generate a sortable unique id
my $reducer = new Gearman::XS::Client;
if ($reducer->add_server('localhost', 4731) != GEARMAN_SUCCESS) {
    printf(STDERR "%s\n", $reducer->error());
    exit(1);
}

my $reducer_id = time . '_' . join '', map { ($a..'z', 0..9)[rand 36] } (1..4);

- Connect the mapper client to the gearman server listening on 4730. More than one can be defined here.
- Connect the reducer client to the gearman server listening on 4731. More than one can be defined here.
- A unique reducer id is generated for each reducer client. This is used to identify the output.
Example: The Master Process
Submitting a Mapper Job For Each Split

- The company IDs are pushed into an array of splits. A jobs hash for both mappers and reducers is defined.
- A background mapper job for each split is submitted to gearman.
- The job handle is a unique ID generated by gearman when the job is submitted. This is used to identify the job in the jobs hash.
- The split is passed to the job as frozen workload data.

```php
# submit jobs with each split to the mappers
my ($ret, $job_handle);
my $jobs = {};
foreach my $split (@splits) {
    # submit a mapper job to be performed by gearman workers
    ($ret, $job_handle) = $mapper->do_background( 'mapper',
        freeze ( {
            # workload
            'split' => int($split)
        })
    );

    # add this to the jobs to be monitored
    if ($ret == GEARMAN_SUCCESS) {
        print “> Begin mapping $split with job_handle=$job_handle
        $jobs->{$job_handle} = {
            mapper => 1, split => int($split), gearman_client => $mapper
        };
    } else {
        printf(STDERR "%s\n", $mapper->error()) and die;
    }

    # sleep for a tenth of a sec
    select(undef, undef, undef, 0.10);
}
```
Example: The Master Process
Job Monitoring: Submitting Reducer Jobs When Mapper Jobs Are Done

```
# watch the mapper processing...
# when one is complete, submit a job to the reducers
while (1) {
    # stop if there are no more jobs
    last unless (keys %$jobs);
    
    # check each job, run reducer when a mapper is done
    foreach $job_handle (sort keys %$jobs) {
        # get the job status from this job's gearman client
        my ($return_val, $is_status_known, $status, $status_num, $status_denom) =
            $jobs->{$job_handle}->{$gearman_client}->{$job_status}($job_handle);
        
        # this job is done
        unless ($running_status) {
            # this is a complete mapper job... run its reducer
            if ($jobs->{$job_handle}->{$mapper}) {
                ...
            }
            
            # delete the done job
            delete $jobs->{$job_handle};
        }
        
        # sleep for a tenth of a sec
        select(undef, undef, undef, 0.10);
    }
}
```

- Mapper and Reducer jobs are monitored inside of a loop.
- The job status is queried from the client using the job handle.
- If the job is completed and if it is a mapper job, a reducer is now run (detailed in the next slide).
- Completed jobs are deleted from the jobs hash.
Example: The Master Process
Detail on Submitting the Reducer Job

- A background reducer job is submitted to gearman.
- The job handle is a unique ID generated by gearman when the job is submitted. This is used to identify the job in the jobs hash.
- The split and reducer_id is passed to the job as frozen workload data.

```php
# this is a complete mapper job... run its reducer
if ($jobs->{$job_handle}->{mapper}) {
    # submit a reducer job to be performed by gearman workers
    ($ret, $re_job_handle) = $reducer->do_background( 'reducer',
        freeze ( [ # workload
            'split' => int($jobs->{$job_handle}->{split}),
            'reducer_id' => $reducer_id
        ]
    )
    );

    # add this to the jobs to be monitored
    if ($ret == GEARMAN_SUCCESS) {
        $jobs->{$re_job_handle} = {
            reducer => 1,
            split => $jobs->{$job_handle}->{split},
            gearman_client => $reducer
        };
    }
}
```
Our AdWords agreements are generally terminable at any time by our advertisers. Google AdSense is the program through which we distribute our advertisers’ AdWords ads for display on the web sites of our Google Network members. Our AdSense program includes AdSense for search and AdSense for content. AdSense for search is our service for distributing relevant ads from our advertisers for display with search results on our Google Network members’ sites.

... Filings Text ...

... Redundant Counts ...

... Aggregated Counts ...

Thursday, December 9, 2010
That’s It For Now
To Be Continued...

Presented By
Jamie Pitts

Thursday, December 9, 2010
Credits

Diagrams and other content were used from the following sources:

MapReduce: Simplified Data Processing on Large Clusters
Jeffrey Dean and Sanjay Ghemawat

Gearman (http://gearman.org)
Brian Aker and Eric Day