MapReduceDelta and Squeal: Running Pig Scripts with Storm

James Lampton
University of Maryland
College Park, MD
jlampton@umd.edu

Jimmy Lin
University of Maryland
College Park, MD
jimmylin@umd.edu

Ashok Agrawala
University of Maryland
College Park, MD
agrawala@umd.edu

ABSTRACT

As developers shift from batch MapReduce to stream processing for better latency, they are faced with the dilemma of changing tools and maintaining multiple code bases. MapReduceDelta is a framework that allows arbitrary chains of MapReduce jobs to be run as pipelined, incremental processes in a stream processing framework. Squeal is a modification of the Pig execution framework that runs lightly modified user scripts on Storm. These modifications center around the specification of input and output locations as well as intermediate state storage. The result is a system that provides sub-second latency for calculations on streams of data.

1. INTRODUCTION

There is an ongoing renaissance in data processing within the field of Computer Science due to the release of MapReduce[5], the availability of large amounts of user generated data, and the availability of large clusters of computers. MapReduce provides an approachable mechanism for programming clustered resources in a manner that compensates for hardware failures. MapReduce is by no means the only mechanism of using large clusters of computers, but it has demonstrated the viability of such solutions and arguably given rise to new methods of using such machines (Spark, BSP, etc.). The advent of the world wide web, social media, and ubiquity of mobile devices has led to an explosion of data. The availability of inexpensive compute resources through various Infrastructure as a Service providers makes these tools available to a broad cross section of users.

MapReduce is designed as a batch processing system. Because of this, its niche is high latency, high throughput calculations. Once a MapReduce job has run, its results can be fed into other systems for quick retrieval of materialized results. Work flow systems can be used to accumulate and schedule jobs based on incoming data. This presents new challenges such as switching to incremental calculation in order to merge in mini-batches of data which may not be possible for all computations.

However, there is a minimal execution latency and associated inefficiencies in using a batch system for continuous computation. Running continuous calculations has long been the purpose of Data Stream Management Systems (DSMS)[4]. Running code developed for a DSMS on a MapReduce cluster generally is not a problem aside from the latency induced by the reduce operators. Running MapReduce code in a DSMS has not generally been possible because the reduce function requires that all the maps be completed before computation can begin. The use of these systems generally requires maintenance of two separate code bases: one for batch (where most algorithms are initially developed) and one for streaming (for low latency responses).

The contribution of this work is MapReduceDelta, a novel framework for converting MapReduce algorithms into pipelined, incremental computations for execution in a streaming environment. Given an abstract data flow, MapReduceDelta can execute it in both batch and streaming with minimal code changes. This is made possible by treating the output of MapReduce operations as message sets which can be used to calculate *delta* messages to propagate side effects through a computation's topology. These capabilities are demonstrated using Squeal, a modification of the Pig[1] scripting language that runs on Storm[2]. This is accomplished by retargeting the Pig physical plan to either Hadoop or Storm as appropriate.

2. RELATED WORK

Recent developments (S4[8], SPADE[6], Storm) have created DSMS systems for clusters of computers. Use of these frameworks requires new development in stream-specific APIs or language constructs. Though streaming code can run on batch systems (S4 and TiMR[2]) or issue batch jobs as necessary (DEDUCE[7]), there has been little work to bring batch codes, such as Hive or Pig, into a streaming environment.

Some have proposed modifications to the MapReduce framework to allow for continuous execution. MapReduce Online[3] uses a modified version of Hadoop to pipeline results between MapReduce phases and jobs. They also mention many of the challenges of shifting algorithms from batch to streaming whereas MapReduceDelta provides a system

[1]http://pig.apache.org
[2]http://storm-project.net/

for propagating side effects through execution pipelines. C-MR[1] uses a custom C++ API based on MapReduce to execute streaming tasks on a multi-processor system. MapReduceDelta aims to allow execution on a single machine or clusters of Hadoop and Storm using the same code base. HStreaming[3] provides an API for developing streaming results in and out of Hadoop processes using data batches in a similar fashion to MapReduce Online. It also provides a construct for windowed results and provide a modified version of Pig that introduces WINDOW keywords for the GROUP operators. Squeal is built to run natively in Storm which allows it to leverage other strengths of the DSMS, such as seamless dynamic interaction with persistent states and better integration with native DSMS codes.

3. SIMPLE EXAMPLE

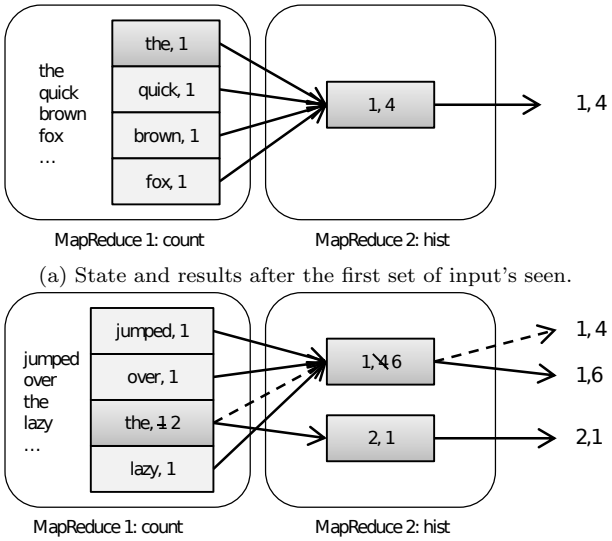
The "Hello World" of MapReduce is "word count." To provide a simple example that illustrates the challenges of moving MapReduce algorithms to a streaming platform, consider the following Pig script that calculates word count followed by the frequency of all word counts:

```
sents = LOAD '$input' AS (sentence:chararray);
words = FOREACH sents GENERATE
    FLATTEN(TOKENIZE(sentence));
lower_words = FOREACH words GENERATE
    LOWER($0) AS word;
count_gr = GROUP lower_words BY word;
count = FOREACH count_gr GENERATE
    group, COUNT(lower_words) AS word_count;
hist_gr = GROUP count BY word_count;
hist = FOREACH hist_gr GENERATE
    group, COUNT(count) AS freq;
STORE hist INTO '$output';
```

This script loads the specified input (*sents*), tokenizes each line (*words*), and generates a lowercase list of words (*lower_words*). This list is then grouped by word and counted (*count*). The next step groups these counts by *word_count* and counts the occurrence of each (*hist*). This data is stored into the specified output location.

Figure 1(a) shows how this script is executed in MapReduce over four unique words "the quick brown fox". Each GROUP statement induces a MapReduce execution. The tables show the accumulated results for each reduce step. Note the highlighted cells representing the word count for "the" (which is one) and the frequency of the word count "1" (which is four).

Figure 1(b) shows what happens when four additional words are considered "jumped over the lazy", resulting in six unique words and one word ("the") occurring twice. To handle the side effects of the updates, the notion of "negative" messages is used to invalidate previous values from each reduce stage. The map functions must be adjusted to appropriately handle these negative messages and propagate the changes through the calculation. In this case, the word count for "the" changes from 1 to 2 which causes a negative message for the previous value, "the, 1" (denoted with a dashed line) and a positive value for the current value, "the, 2". Likewise, the frequency for "1" is decremented once by the negative message from "the" and incremented three



(a) State and results after the first set of input's seen.

(b) State changes and delta messages necessary to after second input.

Figure 1: Two stage MapReduce process to compute the frequencies of word counts.

times by the newly added words. Again, a negative message is emitted for the previous value, "1, 4", and a positive message is transmitted with the new value, "1, 6".

To address these side-effects, the output values from map and reduce stages are given a *sign* marker to represent which type of *delta* message they represent. So called "positive" messages are adding information whereas "negative" messages are invalidating previous values. Propagating these delta messages through the processing topology changes Pig into a streaming computation. Because Pig is more restricted than general Java-based MapReduce, the execution plans can be easily adjusted to accommodate these changes with minimal modification to the user generated script.

4. ARCHITECTURE

Squeal makes use of the Storm clustered streaming environment. Storm was released to the Open Source community in 2011 by Twitter. At the lowest level, Storm has *spouts* (data sources) and *bolts* (user operators) connected by *streams*. A stream is a collection of named channels that route messages containing tuples from spouts to bolts and between bolts. A *topology* is a directed graph of spouts, bolts, and stream configurations that runs on a Storm cluster. Storm clusters are made up of a collection of data center local components. These components include *nimbus*, which assigns the deployment of topologies to *workers* running within *supervisors*. Nimbus is analogous to Hadoop's job tracker, supervisors to Hadoop's task trackers, and workers to Hadoop's task slots. Finally, *Trident* is an API abstraction that implements higher level functionality such as mini-batches and transactional states using Storm much like Cascading[4] does for Hadoop.

Figure 2 shows a model of MapReduceDelta within Trident. The most significant changes to the MapReduce paradigm are the introduction of the delta markers and corresponding pathways in the execution plan. Between the map and

[3]http://www.hstreaming.com

[4]http://www.cascading.org/

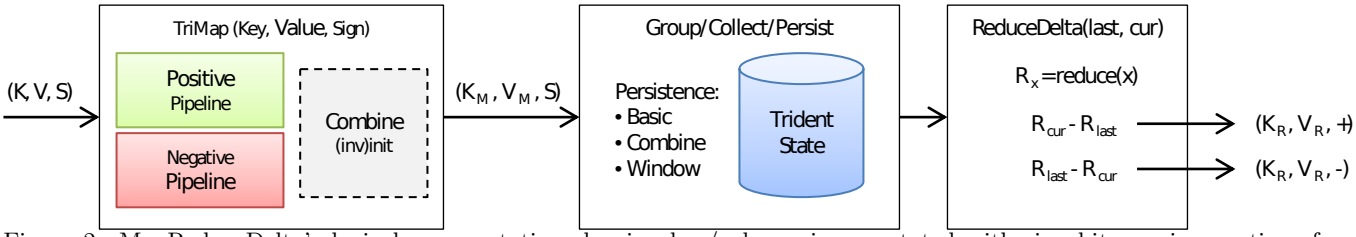


Figure 2: MapReduceDelta's logical representation showing key/value pairs annotated with sign bits, various options for intermediate state management, and formulae for creating delta messages from state changes.

reduce stages lies Trident's state mechanism which is used to accumulate and merge results before passing them to the ReduceDelta operation. In this case, Trident state is a distributed map from keys to values. These key/value pairs can be cached in memory to front a deeper storage mechanism (such as HBase or Cassandra) or configured to provide simple LRU semantics on a fixed memory buffer. The following section explains the concepts of MapReduceDelta by tracking the changes through the provided example.

4.1 Plan Conversion

Pig works by transforming the user specified script into a logical plan, a physical plan, and finally an optimized MapReduce plan which describes a sequence of jobs to execute in Hadoop. Squeal adds a Storm execution mode to Pig which takes the MapReduce plan and converts it to a Storm topology. To explain these modifications, consider the plan to compute the word count frequencies from the previous section without the use of *combiners* (which will be discussed in a later section).

4.1.1 Calculating Word Count

The first MapReduce stage computes the *count* alias. Starting from the *LOAD* function, Squeal uses a User Defined Function (UDF) to wrap a Storm spout. The conversion routines look for these specific UDFs and replace them with spouts in a topology. The map phase generates a stream of words. This is achieved by taking the physical plan from the map operator and placing it within a trident function, *TriMap*, that mimics the behavior of a Hadoop map operator. As tuples arrive, they are attached to the physical plan's *roots* and the computed values are emptied from the *leaves*. A new tuple stream is created from these results with the appropriate key for routing to the reduce operators.

When data arrives at the *Basic Persist* operator the existing value for the specific key is retrieved (from cache or storage) and a counter keyed by the content of the tuple is incremented or decremented based on the positive or negative state of the tuple respectively. The current and previous values of this state are then sent to a *ReduceDelta* operator. This value/counter data structure is then expanded into the value lists that are expected within the reduce operation. The previous value list is used to generate the *last* message set ($R_{last} = reduce(last)$) while the current value list is used to generate the *current* message set ($R_{current} = reduce(current)$). Negative messages are created using the messages from the last set that are not in the current set ($M_{neg} = R_{last} - R_{current}$) and positive messages are created by messages that appear in the current set and not the last set ($M_{pos} = R_{current} - R_{last}$). The output of the first MapReduce task would be the values of *count* which are materialized to disk. Within the streaming conversion, the next MapReduce phase is directly chained to the output of the ReduceDelta operator.

4.1.2 Calculating Frequencies

The map phase for the *hist_gr* calculation is the identity function. What is important to consider is that at this point negative messages may be transmitted from the previous ReduceDelta. The TriMap operator has two physical plans, one for the positive messages and one for the negative messages. Two pipelines are necessary because the map function can be asynchronous, requiring marking of tuples and derived values or cloning the execution pathways. Cloning the execution pathway greatly simplifies the tracking of sign through the computations.

As with the previous MapReduceDelta step, the values are accumulated and new delta messages are produced. Within Pig, the ultimate goal of a script is to produce a side effect in the form of a store function. For Squeal, the intermediate values represented by the grouped states can provide as much value as the final results stream. The intermediate states can be persisted into large key value stores such as HBase or Cassandra for retrospective analysis. Furthermore, the result stream can be fed to notification systems, business systems for action, or to displays for situational awareness.

4.2 Combiners

If the reduce function has the appropriate algebraic properties it can be implemented with a combiner to reduce the amount of intermediate state necessary to compute the result. Many of the base functions provided by Pig implement combiners to improve the performance of these calculations. Within Pig, the *Algebraic* interface marks a function as combinable. This interface requires *init*, *intermed*, and *final* functions. Squeal adds an *invinit* function which replaces the *init* function within the negative map pipeline.

Consider the following Python code to implement an invertible form of *COUNT*:

```
class Count:
  def init(s, t):
    return 1
  def invinit(s, t):
    return -1
  def intermed(s, c1, c2):
    return c1 + c2
  def final(s, c_list):
    return sum(c_list)
```

When a value from a negative message is initialized a negative value is returned. These values mix appropriately

with the positive values to ultimately return the correct value. Within the topology with combiners the *Combine Persist* state maintains the combined intermediate results. For Combine Persist, a single combine state is kept per key instead of a counting set for the Basic Persist case. This results in significant memory and network transmission savings.

Some base functions are not easily inverted. Consider MAX: if a negative message removes the current maximum value, a new value must replace it. This implies that a set of candidates must be maintained to ensure the correctness of the computation. In this case, more state is necessary to compute the results (perhaps the TOPK), in other cases it is best to switch to non-combine reduces and limit the memory requirements using windowing operations or by expanding the grouping key to cause window-like behavior.

4.3 Windowing and Other Considerations

Within stream processing, windowing is used to provide bounds on intermediate state and to make some calculations tractable (such as the aforementioned MIN and MAX). Windowing also provides a means to join two streams of data. Squeal adds window functionality by providing a *Window Persist* operator, which extends the Base Persist functionality. This functionality can be activated by annotating the script with windowing parameters on GROUP or JOIN aliases to describe the window sizes for each of the input streams. Negative messages are disabled following a windowed operator because the streaming side effects are now explicitly handled by the script.

Pig is often used to enrich data by joining against static data sets (such as a data set that maps postal codes to city and state). These joins can be achieved using replicated hash joins in the memory of each mapper or in a distributed fashion using the reducers. To implement this functionality, Squeal analyzes a script for static inputs, dispatches the static parts of the subtree to a Hadoop cluster for execution, and loads the results from HDFS within the map function (for replicated joins) or into the Trident state for reduce-based joins. For slow-changing data sets, this may be all that is necessary. For large enrichment data sets that are not truly static, this provides stopgap functionality to use streaming while the accompanying data set is also switched to streaming computation. The following section describes such a situation.

5. DEMONSTRATION PLAN

The demonstration will present NationMind implemented in Squeal running on an Amazon EC2 cluster. NationMind is a system developed to analyze topic references in Twitter during the 2012 US Presidential Debates. The demonstration shows the batch version of the script running in a Hadoop environment. The script is then modified to read Tweets from a messaging system and executes in Squeal. This demonstrates Squeal's use of Hadoop to process static enrichment data[9], which is then mixed with online streamed data. Tweets collected during various events (the Presidential Debates, Hurricane Sandy, and the 2013 State of the Union address) are fed through the system to show dynamic tracking of response to the events.

6. CONCLUSION

As people shift from batch to interactive and streaming, MapReduceDelta and Squeal demonstrate the ability to have a single code base that can exploit these complimentary execution environments. This gives developers flexibility and significantly lowers the time and effort necessary to progress from prototyping on historical data in a batch environment to executing against live data streams. Ultimately, this should provide a means of shifting from traditional question-based, post-mortem data interaction to an alert/notification based interaction. Being able to react to data as it occurs will have significant impact in public safety, medical, and marketing fields. This could also be used in televised events to address concerns of the national audience.

7. REFERENCES

[1] N. Backman, K. Pattabiraman, R. Fonseca, and U. Cetintemel. C-mr: continuously executing mapreduce workflows on multi-core processors. In *Proceedings of third international workshop on MapReduce and its Applications Date*, pages 1–8. ACM, 2012.

[2] B. Chandramouli, J. Goldstein, and S. Duan. Temporal analytics on big data for web advertising. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 90–101. IEEE, 2012.

[3] T. Condie, N. Conway, P. Alvaro, J. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 21–21, 2010.

[4] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 44(3):15, 2012.

[5] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.

[6] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo. Spade: the system s declarative stream processing engine. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1123–1134, New York, NY, USA, 2008. ACM.

[7] V. Kumar, H. Andrade, B. Gedik, and K. Wu. Deduce: at the intersection of mapreduce and stream processing. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 657–662. ACM, 2010.

[8] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. *Data Mining Workshops, International Conference on*, 0:170–177, 2010.

[9] V. I. Spitkovsky and A. X. Chang. A cross-lingual dictionary for english wikipedia concepts. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC12), Istanbul, Turkey*, 2012.