SkewTune in Action

Mitigating Skew in MapReduce Applications

YongChul Kwon[1], Magdalena Balazinska[1], Bill Howe[1], Jerome Rolia[2]
[1] University of Washington, [2] HP Labs
{yongchul,magda,billhowe}@cs.washington.edu, jerry.rolia@hp.com

ABSTRACT

We demonstrate SkewTune, a system that automatically mitigates skew in user-defined MapReduce programs and is a drop-in replacement for Hadoop. The demonstration has two parts. First, we demonstrate how SkewTune mitigates skew in real MapReduce applications at runtime by running a real application in a public cloud. Second, through an interactive graphical interface, we demonstrate the details of the skew mitigation process using both real and synthetic workloads that represent various skew configurations.

1. INTRODUCTION

We observe that the increased demand for complex analytics has translated into an increased demand for user-defined operations (UDOs) — relational algebra and its close derivatives are not enough [15, 19]. But UDOs complicate the algebraic reasoning and other simplifying assumptions relied on by the database community to optimize execution. Instead developers rely on "tricks" to achieve high performance: ordering properties of intermediate results, custom partitioning functions, extensions to support pipelining [4] and iteration [3], and assumptions about the number of partitions. For example, the Hadoop-based sort algorithm that won the terasort benchmark in 2008 required a custom partition function to prescribe a global order on the data [18]. Moreover, when these UDOs are assembled into complex workflows, the overall correctness and performance of the application becomes sensitive to the characteristics of individual operations.

MapReduce [5] has proven itself as a powerful and cost-effective approach for writing UDOs and applying them to massive-scale datasets [1]. MapReduce provides a simple API for writing UDOs: a user only needs to specify a serial map function and a serial reduce function. The implementation takes care of applying these functions in parallel to a large dataset in a shared-nothing cluster.

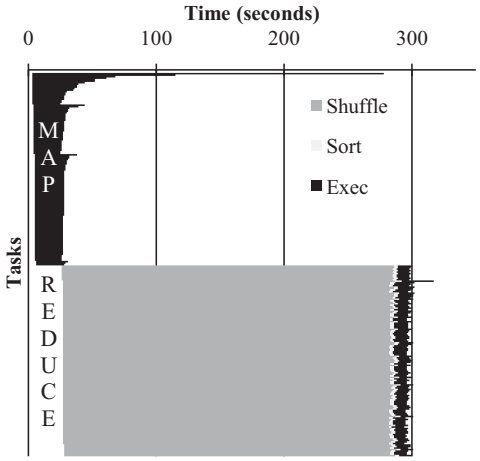


Figure 1: A timing chart of a MapReduce job running the PageRank algorithm from Cloud 9, Exec represents the actual map and reduce operations. The slowest map task (first one from the top) takes more than twice as long to complete as the second slowest map task, which is still five times slower than the average. If all tasks took approximately the same amount of time, the job would have completed in less than half the time.

While MapReduce is a popular data processing tool [1], it still has several important limitations. In particular, skew is a significant challenge in many applications executed on this platform [10, 12, 16]. When skew arises, some partitions of an operation take significantly longer to process their input data than others, slowing down the entire computation.

Figure 1 illustrates the problem. We use PageRank [2] as an example of a UDO[1]. As the figure shows, this UDO is expressed as a MapReduce *job*, which runs in two main phases: the map phase and the reduce phase. In each phase, a subset of the input data is processed by distributed *tasks* in a cluster of computers. Each task corresponds to a partition of the UDO. When a map task completes, the reduce tasks are notified to pull newly available data. This transfer process is referred to as a shuffle. All map tasks must complete before the shuffle part of the reduce phase can complete, al-

[1] `http://lintool.github.com/Cloud9/`

lowing the reduce phase to begin. Load imbalance can occur either during the map or reduce phases. We refer to such an imbalanced situation as *map-skew* and *reduce-skew* respectively. Skew can lead to significantly longer job execution times and significantly lower cluster throughput. In the figure, each line represents one task. Time increases from left to right. This job exhibits map-skew: a few map tasks take 5 to 10 times as long to complete as the average, causing the job to take twice as long as an execution without outliers.

In previous work, we proposed an adaptive skew mitigation technique for MapReduce applications and implemented it in a system that we called SkewTune [13]. SkewTune can handle two common types of skew: (1) skew caused by an uneven distribution of input data to operator partitions (or tasks) and (2) skew caused by some input records (or key-groups) taking much longer to process than others. For these sources of skew, speculative execution, a popular strategy in MapReduce-like systems [5, 9, 11] to mitigate skew stemming from a non-uniform performance of physical machines, is ineffective because the speculative tasks execute the same code on the same data and therefore do not complete in any less time than the original tasks.

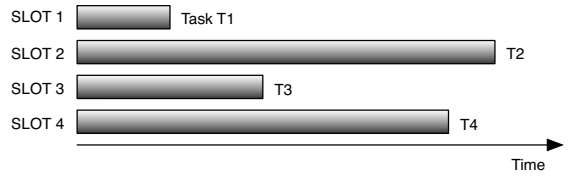
In this demonstration, we show SkewTune's skew mitigation approach through a step-by-step interactive demonstration. We also demonstrate the prototype system running in a cloud and how it helps a real application experiencing skew at runtime.

2. SKEWTUNE: OVERVIEW

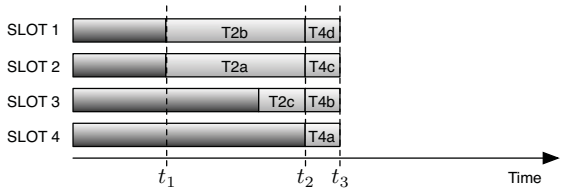
SkewTune takes a Hadoop job as input. For the purpose of skew mitigation, SkewTune considers the map and reduce phases of the job as separate UDOs. In SkewTune, as in Hadoop, a UDO pulls its input from the output of the previous UDO, where it is buffered locally. A UDO is assumed to process its input one *record* at a time with no state preserved between individual input records. A key-value pair (*i.e.*, mapper input) and a key group (*i.e.*, reducer input) are each considered a special cases of records. Each UDO is parallelized into tasks, and each task is assigned a *slot* in the cluster. There is typically one slot per CPU core per node. When a task completes, the slot becomes available.

SkewTune's skew mitigation technique is designed for MapReduce-type data processing engines. The three important characteristics of these engines with respect to skew handling are the following: (1) A coordinator-worker architecture where the coordinator node makes scheduling decisions and worker nodes run their assigned tasks. On completion of a task, the worker node requests a new task from the coordinator. This architecture is commonly used today [5, 6, 9, 11]. (2) De-coupled execution: Operators do not impose back-pressure on upstream operators. Instead, they execute independently of each other. (3) Independent record processing: The tasks are executing a UDO that processes each input record (possibly nested) independently of each other. Additionally, SkewTune requires (4) Per-task progress estimation, t_{remain}, which estimates the time remaining [17, 21] for each task. Each worker periodically reports this estimate to the coordinator. (5) Per-task statistics: each task keeps track of a few basic statistics such as the total number of (un)processed bytes and records.

Figure 2 illustrates the conceptual skew mitigation strategy of SkewTune. Without SkewTune, the operator completion time is dominated by the slowest task (*e.g.*, T2 in Figure 2(a)). With SkewTune, as shown in Figure 2(b), the system detects that T2 is experiencing skew at t_1 when T1 completes. SkewTune labels T2 as *the straggler* and mitigates the skew by repartitioning T2's *remaining* unprocessed input data. Indeed, T2 is not killed but rather terminates early as if all the data that it already processed was the only data it was allocated to process. Instead of repartitioning T2's remaining input data across only slots 1 and 2, SkewTune *proactively* repartitions the data to also exploit slot 3, which is expected to become available when T3 completes. SkewTune re-partitions the data such that all new partitions complete at the same time. The resulting subtasks T2a, T2b, and T2c are called *mitigators* and are scheduled in the longest processing-time first manner. SkewTune repeats the detection-mitigation cycle until all tasks complete. In particular, at time t_2, SkewTune identifies T4 as the next straggler and mitigates the skew by repartitioning T4's remaining input data.



(a) Without SkewTune, the operator runtime is that of the slowest task.



(b) With SkewTune, the system detects available resources as task T1 completes at t_1. SkewTune identifies task T2 as the straggler and re-partitions its unprocessed input data. SkewTune repeats the process until all tasks complete.

Figure 2: Conceptual skew mitigation in SkewTune

To summarize, SkewTune mitigates skew at runtime by repeating the following three steps.

1. **Detect:** The coordinator observes the execution of all tasks in a MapReduce phase and collects their time remaining estimates. When a slot becomes idle, the coordinator identifies the straggler, which is the task with the longest time remaining estimate.
2. **Scan:** The coordinator stops the straggler task. The unprocessed data is scanned either locally or in parallel to collect information for repartitioning. The information is sent to the coordinator.
3. **Plan:** The coordinator plans how to repartition the remaining data of the straggler task using the information from the scan and the time remaining estimates of all running tasks. Once the mitigators are scheduled, the coordinator goes back to the Detect phase and continues.

We refer the interested reader to our previous work for more details about the SkewTune technique and prototype implementation [13, 14].

3. DEMONSTRATION

The demonstration has two key components. The first component shows the prototype SkewTune implementation in action on a real MapReduce application. The second component is an interactive walk-through of the SkewTune approach using both real and synthetic workloads that represent various skew scenarios. We describe both components here.

3.1 SkewTune in the Cloud: Start

Before the demonstration begins, we deploy a four- to eight-node Hadoop cluster running the SkewTune prototype in a public cloud such as Amazon EC2.

When the demonstration starts, we first launch a new Hadoop job in the SkewTune cluster. The goal is to show how the prototype system mitigates the skew that arises in the job at runtime. The application is setup such that the job completes within 5 minutes, which is the duration of the entire demonstration.

For the demonstration, we build a sorted inverted index over a subset of English Wikipedia pages. To get the sorted index, we assume that the user wrote a custom partitioner such that each reduce task processes all the words starting with the same character. There are thus always 27 reduce tasks. The application naturally exhibits skew due to an uneven distribution of the start characters across words in the corpus.

3.2 SkewTune in Action

While the job is running in the background, we present an overview of SkewTune's skew mitigation approach with a poster and an interactive demonstration especially focusing on the key techniques (*e.g.*, late skew detection, proactive repartitioning, and the control flow of skew mitigation) [13].

The goal of the interactive demonstration is to show when skew arises in a MapReduce application and how SkewTune mitigates that skew at runtime step-by-step. The demonstration is a simulation based on real execution traces and real data. We summarize the real data with statistics that include record sizes in bytes and per-record processing times for map tasks. The statistics include the size in bytes and the processing time for each key-group and each value for reduce tasks. We also prepare the following synthetic workloads so that we can interactively control the ordering and the size of the input data as well as the task processing time and, by doing so, explore various skew problems.

- A few reduce tasks are assigned too much data.
- A few map() or reduce() function invocations are significantly more expensive than others either due to input data sizes or specific data values.

For both workloads, we will allow interactive changes to various parameters such as processing time per record, size of each record, the number of records per reduce, processing rate per node so that a skew can be dynamically introduced during the demonstration.

The demonstration simulates the execution of a given workload and the audience sees the details of each step in the skew mitigation process. We briefly describe each of SkewTunes's skew mitigation steps and indicate what the audience will see during this part of the demonstration.

- **Detect:** The coordinator (*i.e.*, JobTracker in Hadoop) monitors and collects time remaining estimates for all running tasks. When a slot becomes idle (*i.e.*, running out of tasks to schedule), the coordinator chooses the straggler as the task with the longest time remaining estimate. The demonstration will show the running tasks with their estimates derived from execution traces and will highlight the chosen straggler task once SkewTune identifies it.
- **Scan:** The coordinator sends a message to the straggler task. The task stops and scans its remaining input data to collect information for repartitioning that data. The scan is performed either locally or in parallel depending on the estimated cost. The demonstration will show how the scan proceeds and the information that is collected.
- **Plan:** The coordinator plans how to repartition the straggler's remaining input data based on (a) the information collected during the scan and (b) the current time-remaining estimates of all running tasks. The demonstration will show the key steps of this planning phase and the resulting assignment of input data to each mitigator task.
- **Repeat:** The mitigator tasks are scheduled, the demonstration goes back to the Detection phase, and the above steps repeat until the job terminates.

The interactive demonstration runs as a discrete event simulation. It will thus run faster than the wall clock and will also have the capability to pause. We will thus be able to run the demonstration multiple times using various workloads, comparing with the executions without SkewTune, and pause the system at any time to answer questions and discuss any system details of interest to the audience. We will also allow interactive changes to various parameters such as processing time per record, processing rate per node so that the audience can see how such dynamically introduced skew gets mitigated away by the system.

3.3 SkewTune in the Cloud: Wrap-up

By the time the interactive demonstration ends, the job running in the cloud will complete. We will show how the prototype SkewTune system mitigated the skew at runtime by showing the Hadoop job tracker web interface and a task scheduling timing chart as shown in Figure 1. We will then conclude the demonstration by answering any follow-up questions.

4. RELATED WORK

Handling load imbalance has also been studied previously in the context of MapReduce applications. In earlier work, we proposed SkewReduce, a system that statically optimizes the data partitioning according to user-defined cost functions [12]. The approach effectively addresses potential data skew problems, but it relies on domain knowledge from users and is limited to specific types of applications. Ibrahim *et al.* and Gufler *et al.* studied data skew in the reduce phase [7, 10]. Both approaches schedule reduce keys to the reduce tasks based on cost models. Also, the reduce key scheduling does not preserve the order as in the original reduce output. SkewTune not only addresses skew in both the map and reduce phases but also minimizes the side-effect of skew mitigation by preserving input order.

In the follow-up work, Gufler *et al.* proposed the TopCluster approach to construct a histogram of all reduce keys to

identify skewed reduce keys [8]. The TopCluster approach is similar to reconciling the result of parallel scan in SkewTune [14]. TopCluster eagerly monitors, detects, and mitigates reduce skew while SkewTune lazily detects and mitigates skew. Also, in SkewTune, the planning is done using exact information if possible.

Vernica *et al.* proposed an adaptive MapReduce system using situation-aware mappers [20]. The situation-aware mappers continuously monitor the execution of mappers and adaptively resplit the map input data. Also, with an adaptive combiner and partitioner, the system also tries to balance the reduce input. However, the situation-aware mappers can not handle computational skew at the reducers, where some key-groups take longer to process than others.

Overall, the above eager approaches are complementary to SkewTune. SkewTune can handle anything left by each of the above systems in a transparent manner.

Acknowledgments

We thank the anonymous reviewers for their helpful comments on early drafts of this paper. This work is supported in part by the National Science Foundation CAREER grant IIS-0845397, the UW eScience Institute, and an HP Labs Innovation Research Award.

5. REFERENCES

[1] Apache Hadoop Project. Powered By Hadoop. `http://wiki.apache.org/hadoop/PoweredBy/`, 2011.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. of the 7th WWW Conf.*, pages 107–117, 1998.

[3] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst. HaLoop: Efficient iterative data processing on large clusters. *Proc. of the VLDB Endowment*, 3(1):285–296, 2010.

[4] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce online. In *Proc. of the 7th NSDI Symp.*, 2010.

[5] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *Proc. of the 6th OSDI Symp.*, 2004.

[6] D. J. DeWitt, E. Paulson, E. Robinson, J. Naughton, J. Royalty, S. Shankar, and A. Krioukov. Clustera: an integrated computation and data management system. *Proc. of the VLDB Endowment*, 1(1):28–41, 2008.

[7] B. Gufler, N. Augsten, A. Reiser, and A. Kemper. Handling data skew in mapreduce. In *The First International Conference on Cloud Computing and Services Science*, pages 574–583, 2011.

[8] B. Gufler, N. Augsten, A. Reiser, and A. Kemper. Load balancing in mapreduce based on scalable cardinality estimates. In *Proc. of the 28th ICDE Conf.*, 2012.

[9] Hadoop. `http://hadoop.apache.org/`.

[10] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi. LEEN: Locality/Fairness-Aware Key Partitioning for MapReduce in the Cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 17–24, 2010.

[11] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proc. of the EuroSys Conf.*, pages 59–72, 2007.

[12] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. Skew-resistant parallel processing of feature-extracting scientific user-defined functions. In *Proc. of the First SOCC Conf.*, pages 75–86, June 2010.

[13] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. SkewTune: Mitigating Skew in MapReduce Applications. In *Proc. of the SIGMOD Conf.*, pages 25–36, 2012.

[14] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. SkewTune: Mitigating Skew in MapReduce Applications. Technical Report UW-CSE-12-03-03, University of Washington, March 2012.

[15] Y. Kwon, D. Nunley, J. P. Gardner, M. Balazinska, B. Howe, and S. Loebman. Scalable clustering algorithm for N-body simulations in a shared-nothing cluster. In *Proc. of the 22nd Scientific and Statistical Database Management Conference (SSDBM)*, pages 132–150, 2010.

[16] J. Lin. The Curse of Zipf and Limits to Parallelization: A Look at the Stragglers Problem in MapReduce. In *7th Workshop on Large-Scale Distributed Systems for Information Retrieval*, number July, 2009.

[17] K. Morton, A. Friesen, M. Balazinska, and D. Grossman. Estimating the progress of MapReduce pipelines. In *Proc. of the 26th ICDE Conf.*, pages 681–684, Mar. 2010.

[18] O. O'Malley. Apache hadoop wins terabyte sort benchmark. `http://developer.yahoo.com/blogs/hadoop/posts/2008/07/apache_hadoop_wins_terabyte_sort_benchmark/`.

[19] The Mahout Team. Apache mahout project. `http://mahout.apache.org/`.

[20] R. Vernica, A. Balmin, K. S. Beyer, and V. Ercegovac. Adaptive MapReduce using situation-aware mappers. In *EDBT*, pages 420–431, 2012.

[21] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving MapReduce performance in heterogeneous environments. In *Proc. of the 8th OSDI Symp.*, 2008.