Map-Reduce Meets Wider Varieties of Applications

Shimin Chen, Steven W. Schlosser

IRP-TR-08-05

Research at Intel

Map-Reduce Meets Wider Varieties of Applications

Shimin Chen
Intel Research Pittsburgh
4720 Forbes Ave, Suite 410
Pittsburgh, PA 15213, USA
shimin.chen@intel.com

Steven W. Schlosser
Intel Research Pittsburgh
4720 Forbes Ave, Suite 410
Pittsburgh, PA 15213, USA
steven.w.schlosser@intel.com

ABSTRACT

Recent studies and industry practices build data-center-scale computer systems to meet the high storage and processing demands of data-intensive and compute-intensive applications, such as web searches. The Map-Reduce programming model is one of the most popular programming paradigms on these systems. In this paper, we report our experiences and insights gained from implementing three data-intensive and compute-intensive tasks that have different characteristics from previous studies: a large-scale machine learning computation, a physical simulation task, and a digital media processing task. We identify desirable features and places to improve in the Map-Reduce model. Our goal is to better understand such large-scale computation and data processing in order to design better supports for them.

1. INTRODUCTION

World data is growing exponentially, doubling its size every three years [8]. Huge amounts of data are being generated from digital media (images/audio/video), web authoring, scientific instruments, physical simulations, and so on. Effectively storing, querying, analyzing, understanding, and utilizing these immense data sets presents one of the grand challenges to the computing industry and research community.

Recent studies and industry practices [5, 6, 13] build data-center-scale computer systems to meet the high storage and processing demands of a sample of these applications, such as web searches. Such a system is composed of hundreds, thousands, or even millions of commodity computers connected through local area networks housed in a data center. It has much larger scales than a traditional computer cluster (typically of tens of machines), while enjoying better and more predictable network connectivity than wide-area distributed computing.

One of the most popular programming paradigms on data-center-scale computer systems is the Map-Reduce programming model [5]. Under this model, an application is implemented as a sequence of Map-Reduce operations, each consisting of a Map phase and a Reduce phase that process a large number of independent data items. The system supports automatic parallelization, distribution of computations, task management, and fault tolerance in hope that programmers can focus on application algorithms without worrying about these complex issues.

Encouraged by its previous successes, we would like to exploit Map-Reduce for a wider varieties of data-intensive and compute-intensive applications. In particular, we have implemented several application tasks, including a large-scale machine learning computation [14], a physical simulation task [1], and a digital media processing task [12]. We used Hadoop [11], an open-source Java-based implementation of the Map-Reduce model. Our goal is to understand the application needs, the strengths and weaknesses of Map-Reduce, and gain insights on how to effectively support data-intensive and compute-intensive applications.

In this paper, we report our preliminary findings. Interestingly, we find Map-Reduce is not an exact fit for supporting these algorithms. Some features (such as automatic parallelization, task distribution, and fault tolerance) are very desirable, while others (such as the Reduce phase) may not be needed. Moreover, our findings suggest that optimization strategies or hints are very desirable. We describe several potential ways to improve upon this.

The rest of the paper is organized as follows. Section 2 describes the Map-Reduce programing model, system services, and advanced features. Section 3 presents our studies of implementing three application tasks on a 400-core computer cluster. Section 4 summarizes the insights learned from the application studies and discusses implications for systems and data management support for data-intensive and compute-intensive applications. Section 5 presents related work and Section 6 concludes the paper.

2. UNDERSTANDING THE MAP-REDUCE PROGRAMMING MODEL AND SYSTEM

When talking about Map-Reduce, one may mean the Map-Reduce programming model (the narrow sense), or the Map-Reduce system (the wider sense) that provides the support for the programming model, as well as a rich set of system services and advanced features. Understanding the programming model is sufficient to write a *semantically correct* program. However, from our experiences, in order to write a program that *performs well* on a data-center-scale system, it is often necessary to understand the Map-Reduce system. In the following, we describe the programming model (Section 2.1), system services (Section 2.2), and advanced features (Section 2.3), based on the original Map-Reduce paper [5] and the open source Hadoop implementation [11].

2.1 Programming Model

Figure 1 illustrates the Map-Reduce programming model using a word counting example. A programmer implements two functions, a Map function and a Reduce function. Their semantics are shown at the top of Figure 1. Conceptually, the input to the Map-Reduce computation consists of a list of (in_key, in_value) pairs. Each Map function call takes a pair of (in_key, in_value) as input and produces zero or more pairs of intermediate key-value, (mid_key, mid_value). Then, the system automatically performs a group-by operation on the intermediate mid_key. After that, each Reduce function call processes a group, i.e. a mid_key and a list of mid_values, and produces zero or more output results. In the word counting example shown in Figure 1, the Map function generates a $(word, 1)$ as an intermediate result for each encoun-

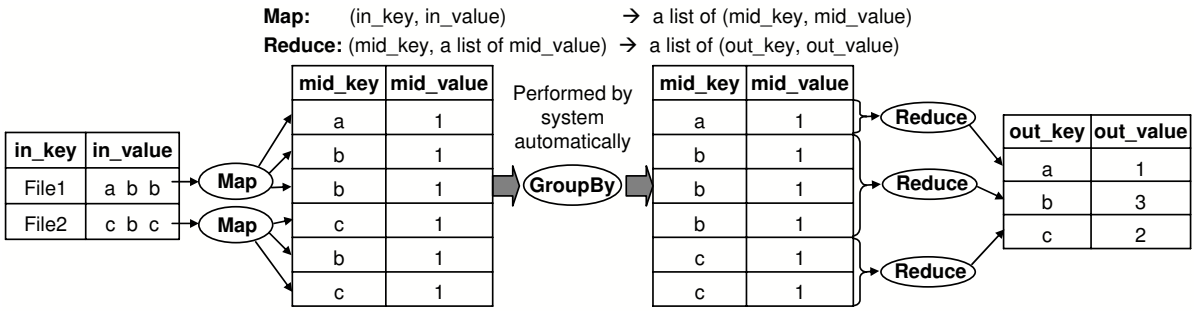


Figure 1: The Map-Reduce programming model illustrated with a word counting example.

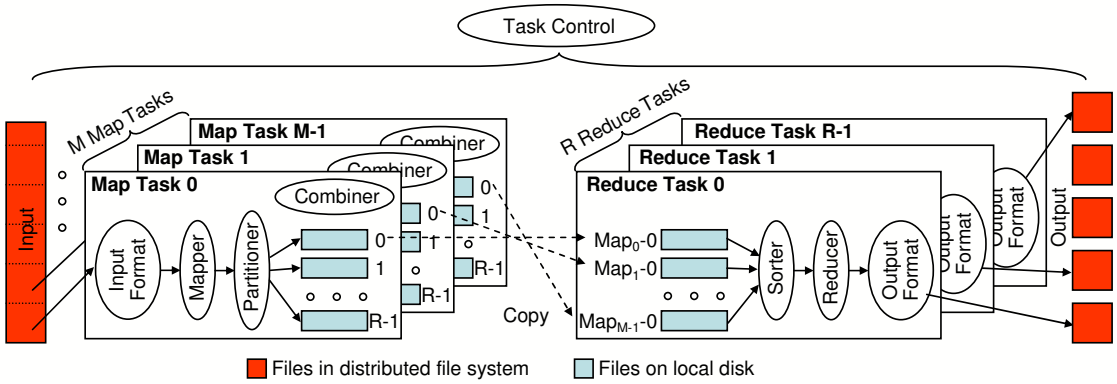


Figure 2: The Map-Reduce system automatically distributes M Map tasks and R Reduce tasks across a large number of computer nodes. (M and R are specified in the Map-Reduce job configuration).

tered word, where $mid\_value = 1$. The Reduce function simply sums up the list of mid_values to obtain the word count.

2.2 System Services

The system provides three categories of services to facilitate the running of a Map-Reduce program across a large number of machines, as shown in Figure 2. **(i) Distributed file system**: the input and output are stored in a distributed file system to be globally accessible. **(ii) Automatically distributing tasks**: the system instantiates Map and Reduce tasks across a large number of machines, partitions and assigns the input to individual Map tasks, then coordinates the copying of intermediate results (stored on local disks) from the machines running Map tasks to machines running Reduce tasks. A typical optimization is to co-locate a Map task to a machine (or rack) that also holds a copy of the assigned input partition in the distributed file system. **(iii) Fault tolerance**: the system monitors the task executions and re-executes tasks if they fail (due to machine crashes etc.). Since the input to a Reduce task may come from any of the Map tasks, it is necessary to wait for all Map tasks to finish before launching Reduce tasks. Near the end of the Map or the Reduce phase, the system schedules backup executions of the remaining in-progress tasks to protect against "stragglers", which are machines taking an unusually long time to complete tasks due to permanent or transient hardware or software problems.

2.3 Advanced Features

Figure 2 shows the many advanced components surrounding and supporting the Mapper and the Reducer computations. Typically, the default implementations of these components can be replaced with customized implementations. **InputFormat** extracts the input record (in_key, in_value) from the input (default: extracting a line from a text file). This gives part of the functionality of a database access method. **Partitioner** computes a Reduce task ID from mid_key, determining which local file to append an intermediate result (default: hash code(mid_key) modulo R). One may perform range based partitioning instead. The local files containing intermediate results are copied to the machine running the corresponding Reduce task. Then **Sorter** sorts the intermediate results to complete the group-by operation (default: merge sort). After the reduce computation, **OutputFormat** formats output results for writing to output files (default: out_key tab out_value per line). The network traffic of copying can be reduced by implementing **Combiner**, which is a partial reduce function, to be called on the intermediate results local on individual Map task machines. This is an optimization for reduce operations that are associative and commutative. For example, in the above word counting example, we can use the Reduce function as the Combiner.

3. APPLICATION STUDIES

We are actively exploiting data-center-scale computing for supporting data-intensive and compute-intensive applications. There is a cluster of 50 blade servers in Intel Research Pittsburgh. Each server contains two 2.33GHz quad-core Intel Xeon E5345 CPUs with 8GB of RAM and 300GB of local disk space running 64-bit Linux and Hadoop 0.15.0. The blade servers are connected through 1Gb Ethernet. Altogether there are 400 CPU cores in this cluster. We collaborate with researchers who are working on computer vision, machine learning, machine translation, scientific simulations, and speech processing. In the following, we describe three of our ongoing efforts for exploiting data-center-scale computer systems to support large-scale data-intensive computations.

3.1 A Machine Learning Task

The first application is computing per-user-pair contingency tables for the Netflix Prize data. Netflix is an online DVD rental com-

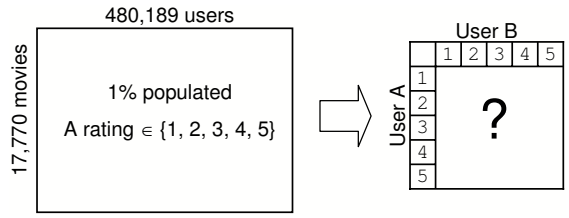


Figure 3: Computing contingency table per user pair for the Netflix Prize training set.

pany, whose customers can give 1-5 integer ratings to the movies. Netflix would like to utilize accumulated rating data to predict the user preferences of movies. That is, when a user is browsing a movie description web page, Netflix will show a predicted rating of the user for the movie. In Oct 2006, Netflix started a contest open to any one for the best algorithm that beats Netflix's own algorithm for predicting ratings by at least 10% [14]. By June 2007, over 20 thousand teams had registered for the competition from over 150 countries, and 2 thousand teams had submitted over 13 thousand prediction sets [2]. The current best submission achieves 9.08% improvement [14].

The contest provides a training data set, which includes 17,770 movies and 480,189 users. For each movie, there is a rating file that contains a list of $(usrID, rating, date)$ records. As shown in the left part of Figure 3, the data set can be viewed as a large sparse matrix that are stored in row order, where rows are movies, columns are users, and matrix elements are ratings (ignoring dates for simplicity). About 1% of the ratings in the matrix are given in the training data set. The goal of the contest is to generate predictions for a list of movie-user pairs whose ratings are only known to Netflix. Therefore, Netflix is able to compute an accuracy score for every submission.

We chose to use the contest as a data-intensive and compute-intensive application and are exploring several machine learning algorithms. One algorithm requires the computation of per-user-pair contingency tables, as shown in the right part of Figure 3. In the contingency table for user A and B, the element at the i-th row and j-th column is the number of movies that A rated i while B rated j. Therefore, the contingency table shows the correlation between A's ratings and B's ratings. For example, if the diagonal elements are much larger than the rest, then A's rating is a good predictor for B's rating and vice versa. We find that over 87% of the contingency tables are non-empty because a few popular movies were rated by almost every user. Therefore, this computation generates contingency tables for over 100 billion different user pairs, which is about 5TB of uncompressed data (assuming a 16-bit integer for each contingency table element).

Solution 1. Naive Map-Reduce. At first glance, this computation looks similar to the previous word counting example, where the "word" here is "userA&userB". We implemented a Map function that generates $(userA\&userB, ratingA\&ratingB)$ for each pair of user A and B ($A < B$) in a given movie file. After the automatic group-by, each group contains all the information for a pair of users. Therefore, the Reduce function simply computes a contingency table from every group of intermediate results. However, the performance of this solution is very poor. When tried on the entire training set, the Map-Reduce task failed because the local disk capacity was exhausted storing intermediate results. For testing, we excluded the 5% most popular movies from the data set, which reduces the non-empty contingency tables from 87% to about 15%. This much smaller computation took over 4 days to finish using the entire cluster.

There are two major problems of this approach. First, the intermediate data size is huge (2.8 trillion intermediate records, tens of TB of data) compared to word counting, incurring high local disk space and network communication overhead. Combiner is not effective because the probability of finding local records of the same user pairs is low. Second, the computation and disk space requirements of Map tasks are very skewed: A popular movie can have thousands of times more users than an unpopular movie, thus producing millions of times more intermediate records. The Map-Reduce system interpreted the longest-running tasks as stragglers, and executed backup tasks, wasting computation resources.

Solution 2. Transpose + Reduce-less Computation. Realizing the above problems, in the second solution, we transpose the conceptual sparse matrix so that the ratings of every user are contiguous. In this way, we can easily retrieve all the ratings for a given user. Then it is straightforward to retrieve the ratings for a given pair of users and compute their contingency table all on a single machine. For the best performance, we use a C implementation and a job scheduling system to submit the jobs to run on the cluster. Using 50 cores, the computation for the entire data set completed in 20 hours, which is dramatically faster than solution 1.

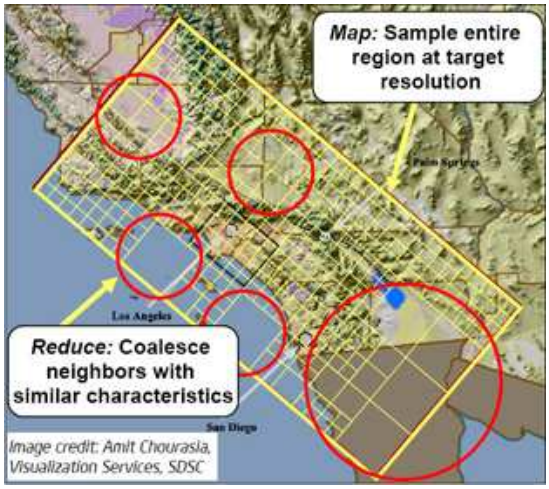
However, the job scheduling system does not support automatic input partitioning, therefore we have to manually split the input. In this respect, it is desirable to implement a Map-Reduce program that does not have the Reduce function. (We call such a program Reduce-less.) The Map function performs the contingency table computations, while the system provides automatic input partitioning and the other useful system services. However, a problem arises because of automatic input partitioning and assignments. Since the computation creates an output file per Reduce task, it is non-trivial to determine the mapping from user pairs to output files and locations in the files. Therefore, it is desirable to automatically create an index structure on the output results.

3.2 A Scientific Simulation Task

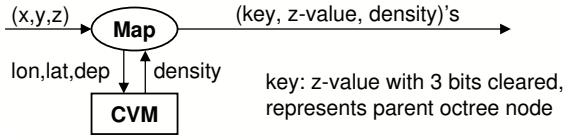
Ground motion modeling, a.k.a. earthquake simulation, models the effects of earthquakes on populated areas for understanding the earthquake energy propagation, analyzing ground motions, and predicting the effects of earthquakes on buildings [1]. This is a very data-intensive and compute-intensive application. Traditionally, it is performed on expensive high-performance supercomputers. We would like to study how well a data-center-scale system with commodity computers (in particular a Map-Reduce system) can support such computations. Currently, we focus on the first computation step in this application, namely building the physical ground model, which are then inputs to later steps.

The basic building block of this computation is a Community Velocity Model (CVM). A CVM is a program that contains observed data (e.g., for seismic events) and uses interpolation to estimate the ground characteristics (e.g., density) for a given location (specified by longitude, latitude, and depth). There are two CVM implementations (SCEC and Harvard). We use the faster (Harvard) CVM implementation here. The task for building a ground model is to query the CVM for all points in a 3D mesh, coalesce neighbors that have similar characteristics, and build an octree structure[1], where each leaf node is a region of points having similar characteristics. Note that coalescing reduces the number of octree nodes, which are proportional to the amount of computation in the rest of the simulation steps. In this study, we generate the ground model for a 600km long x 300km wide x 100km deep region in southern California.

[1]In an octree, the root node represents the entire 3D region. A non-leaf node has (up to) 8 child nodes, dividing its region into 8 disjoint equal-sized octants. Octrees are 3D versions of quadtrees [9].



(a) Overview of the Map-Reduce computation.



(b) The Map function and the manipulated keys.

Figure 4: Building a ground model for southern California.

Solution 1. Map-Reduce with Manipulated Intermediate Keys. The overall Map-Reduce computation is shown in Figure 4(a). The Map function queries the CVM to obtain ground characteristics for each point in the 3D mesh at the target resolution. The Reduce function coalesces neighbors with similar ground characteristics for one level in the octree. After that, we feed the output of the Reduce function to the input of another Map-Reduce computation, where the Map is the identity function and the Reduce function is the same as before, coalescing the next level in the octree. This process is repeated until there is no coalescing or we reach a threshold octree node size T.

Figure 4(b) illustrates the Map implementation. Given a (x, y, z) coordinate, the Map function converts it into $(longitude, latitude, depth)$ and queries the CVM for density information. It computes a Z-value and outputs the Z-value and density as the intermediate value. (Later on, the octree is to be stored into a single-dimensional B-tree structure using the Z-order space filling curve.) In order to allow coalescing at the Reduce function, the Map function generates an intermediate key that represents the parent octree node for this Z-value (by clearing three bits in the Z-value). In this way, after the automatic group-by, a group contains the density information for all the points in a corresponding octree node at one level above. The Reduce function can then decide which node to coalesce based on the similarity of densities. We find this computation took orders of magnitude more time to complete than solution 2.

Solution 2. Generating Samples in Sorted Z-value Order. We made the observation that we can control the order that (x, y, z) samples are generated. If we generate the samples in Z-value order, then the neighbor points in octree nodes will be adjacent to each other. In this way, we can avoid the group-by step in Map-Reduce. We implemented a C program that maintains the sampled locations in a last-in-first-out stack. If a location has similar characteristics as the stack top, then we push it onto the stack. If eight contiguous entries from the top of the stack hold all the children

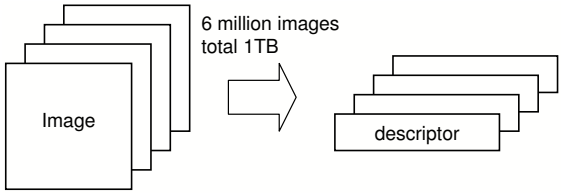


Figure 5: Computing a descriptor (or a set of features) per image for 6 million images. The total input size is about 1TB.

of an octree node, then we coalesce them into a single entry representing the node and put it on the stack top. If a location has different characteristics from the stack top, then we output all the entries in the stack and clear the stack. This implementation builds the ground model in 1 hour using 80 cores in the cluster.

It is desirable to implement a Reduce-less version of the C program that utilizes the Map-Reduce systems service, such as input partitioning and assignment. We divide the input region into sub-regions that have the threshold size T. Each Map task is responsible for generating the octree nodes for a given sub-region. Runtime is comparable to the C implementation, only about 30% slower.

3.3 An Image Processing Task

The third application is a large-scale image processing task, whose goal is to estimate geographic information (where is this scene?) given an image [12]. The idea is to leverage a dataset of over 6 million GPS-tagged images and use scene matching to find out for the given test image the most similar known image(s). Specifically, a set of features (including color, texture, line, etc.) along with the GPS data are extracted from each of the 6 million images and stored in a database. For a test image, the same set of features are extracted and compared against the database for the closest matches. Here, we are concerned with the first step in this application, namely computing a set of features per image for all the 6 million images.

Solution: Reduce-less Computation. Examining the computation, we find that the processing of an image is independent of the processing of the other images. Therefore, our solution simply computes the features for images in parallel. The processing of each image takes about 15 seconds. Using the entire cluster of 400 cores, the computation completed in over 3 days. Like the previous two examples, it is desirable to utilize Map-Reduce system services for this computation. A Reduce-less program can achieve this.

4. IMPLICATIONS FOR SYSTEMS AND DATA MANAGEMENT SUPPORT

We gained deeper understanding of the Map-Reduce systems through the above application studies. Beside these experiences, we believe it is valuable to compare Map-Reduce with the relational data model. In the following, we first perform this comparison to draw a bigger picture of the design space, and then describe the implications for systems and data management support in a data-center-scale computer system.

4.1 The Design Space

In the design space for supporting large-scale parallel computation, there is a trade-off between the flexibility of the programming model and the automatic optimization that the system can support. At one end is the model in a relational database system [16]. All the individual data processing operations are well defined and have multiple alternative implementations in the system. Application tasks are specified in high-level descriptive languages. The system has very good knowledge about the datasets and computations,

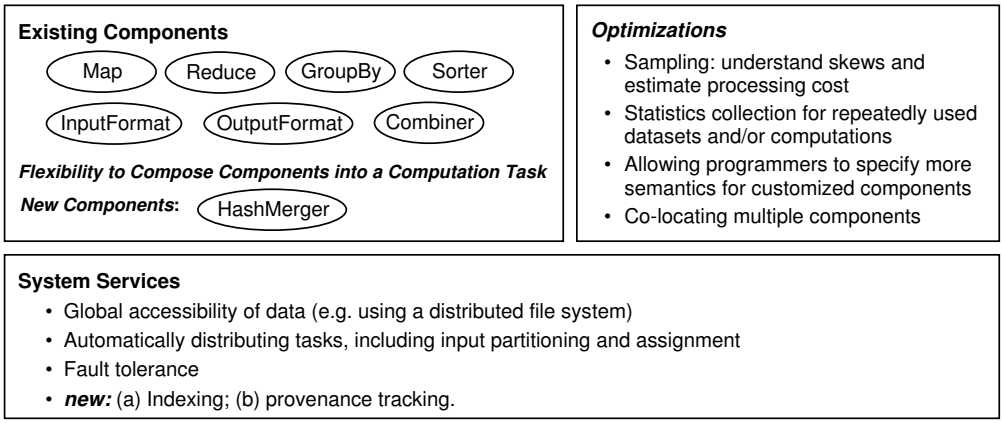


Figure 6: Proposal for enhancing the Map-Reduce system for supporting wider varieties of applications.

thus is able to perform powerful cost-based optimizations. However, the limitation of such a model is its restriction on the datasets and computations. At the other end is manually written distributed parallel programs (e.g., with MPI [17]). There is little restriction on the datasets and the kind of processing that a program can do. However, the underlying system has very limited knowledge about the computations and datasets. Typically, it only knows general resource usages and can perform load balancing type of optimizations. Semantics-specific optimizations are not possible and the supported system services are also very limited.

Map-Reduce is a design point in between the two ends. It imposes a few structures on the datasets (e.g., input key-value pairs are independent) and the computation (i.e., Map-Groupby-Reduce). In this way, it successfully relieves programmers from many complex issues, such as input data assignment, task distributions. At the same time, programmers still have fair amount of flexibility to express customized computations inside the Map and Reduce functions, as well as the other advanced customizable components. Note that the design space in between the two ends is important as evidenced by work in the database community on object-relational and semi-structured data models and the industrial adoption of the Map-Reduce model.

In the following, we discuss implications and potential enhancements as summarized in Figure 6. Essentially, we are trying to cover more design space in between the two ends so that the system can provide effective support for a wider varieties of applications.

4.2 Desirable Systems Services

As shown in Figure 6, the Map-Reduce system services are all very desirable. In our application studies, we consider Reduce-less Map-Reduce programs because the system services significantly simplifies data and task management. Besides the existing system services, we find two additional services are desirable. **(i) Indexing**. Computed results typically reside in multiple files. There is often a need to look up a single result for follow-on computations, which is true for all the three applications. Therefore, an automatic/configurable indexing service could significantly simplify this task. Moreover, the current input scheme is based on a scan of the entire input file(s). If a small fraction of the input data is actually needed, then an indexing scheme can significantly speed up the processing (which is conventional wisdom in the database community). **(ii) Provenance Tracking**. It is often necessary to track the dependence and linkage information of data (e.g., for verification of scientific results). For relational databases, fine-grain tracking data item provenance can be effectively supported because the system knows about the semantics of individual database operations [3]. In contrast, many computations use monolithic programs; it is less feasible to track fine-grain data-item provenance. A Map-Reduce program relieves this problem because each individual data item is processed independently. Therefore, we can potentially support effective fine-grained provenance tracking.

Note that all these system services only assume that the datasets are composed of independent key-value pairs or independent groups of key-value pairs. They do not rely on the Map-Reduce program structure. Therefore, we can enhance a more conventional job management system (such as Condor or Maui/Torque) with the services to achieve a design point that is more flexible than Map-Reduce.

4.3 Flexible Compositions of Components

We find that applications using the Map-Reduce program are limited by the Map-Reduce structure. For example, we would like to use multiple Reduce steps in the solution 1 of ground model building. However, we are forced to use the identity Map function with the Reduce steps, and hence the copying step is always performed. It would be helpful if the system allows flexible composition of components. First, components can be turned off to achieve good performance. For example, the Map component can be disabled. If the input is already sorted, the group-by component can be also disabled. Second, it is desirable to support multiple Map functions reading from different input source files to connect to the same Reduce function (e.g., for joining the inputs).[2] An advantage of explicitly specifying multiple Map functions is that the system gains more knowledge of the computation and potentially performs better optimizations (e.g., on co-locating tasks).

Moreover, the default group-by uses merge-sort to produce groups. However, the database literature indicates that hash-based group-by in many situations are faster [10], therefore it is desirable to support a new **HashMerge** component.

4.4 Optimization Strategies and Hints

Optimizations are highly desirable especially for long-running tasks. In our application studies, we have seen orders of magnitude differences for multiple solutions to the same problem. The obvious question here is how to optimize? As discussed in Section 4.1, the power to optimize is limited by the knowledge of the system about a particular computation task. Therefore, inspired by optimization techniques in the relational databases, we propose the following approaches.

[2]Currently, for supporting joins, one has to hack the Map-Reduce model to identify input source types in the Map function and attach a type tag to the intermediate value.

Sampling. The system can sample a random set of input data items to learn about task characteristics before starting the full-scale run. For example, by sampling, the system should be able to discover the large skews in the solution 1 of the machine learning task. Then it can send a notification to the job submitter, and take more cares in starting backup tasks. The system can also estimate the task runtime and whether local disk space is sufficient.

Statistics Collection. The same datasets may be used repeatedly for multiple slightly different computation tasks. The same kind of computation may be applied to slightly different datasets. For example, an actual movie rating prediction system will need to periodically incorporate new ratings and perform the same computation on the slightly different rating database. For these datasets and computations, it is a good idea to collect statistics for one run and utilize the statistics for improve the performance of following runs.

More Semantics From Programmers. In addition to learning the characteristics of datasets and computations, the system may also provide interfaces that programmers can specify certain useful attributes for datasets and computations. For example, the ordering (or sorted-ness) of a dataset and a computation component. Similar to the database optimization, if the system knows that the input the group-by is sorted, then the group-by step can be skipped.

5. RELATED WORK

Since Google researchers Dean and Ghemawat proposed the Map-Reduce model for data-center-scale computing [5], it has received much attention from across the computing industry and research community. Yahoo has been sponsoring the open-source Java implementation of the Map-Reduce system, Hadoop [11]. Several companies announced plans to make computing resources available for select universities for teaching the Map-Reduce programming model, as well as utilizing it for research purposes.

Previous work exploited the Map-Reduce programming model for parallelizing typical machine learning algorithms on a single machine with multiple processors or cores [4]. The major algorithmic structure to parallelize is the summation form in machine learning algorithms. An example is computing the sum of a large number of values. This work employed multiple Map tasks for computing partial sums, then used a Reduce task for generating the final sum from the partial results. In these computations, the group-by step is unnecessary, supporting our proposal to make the components of a Map-Reduce systems flexibly composable.

Several research studies also aim to improve the Map-Reduce programming model. Yang et al. [18] proposed to add a Merge component after the Reduce function for performing a join operation on two datasets generated by multiple Reduce functions. Isard et al. [13] proposed to allow programmers to specify a DAG for the computation. The DAG specification is sophisticated and a manual DAG representing an SQL query on an astronomy database was illustrated. Olston et al. [15] proposed a SQL-style declarative language, Pig Latin, on top of the Map-Reduce model. All these works tried to bridge the gap between Map-Reduce and the relational databases. In contrast, this paper proposes several enhancements to Map-Reduce based on application studies in order to support wider varieties of applications. We consider design points that are more flexible than Map-Reduce (as discussed in Section 4.2) and as well as design points that provide better optimizations.

DeWitt recently posted an online blog [7] comparing Map-Reduce and parallel relational databases. As discussed in Section 4.1, we believe Map-Reduce and the relational model represent two design points in the design space. It is desirable to come up with designs that can cover the entire design space. This paper and previous works [13, 15, 18] all aim at achieving part of this goal.

6. CONCLUSION

In this paper, we studied three data-intensive and compute-intensive applications that have very different characteristics from previous reported Map-Reduce applications. We find that although we can easily implement a semantically correct Map-Reduce program, achieving good performance is tricky. For example, a computation that looks similar to word counting at the first sight may turn out to have very different characteristics, such as the number and variance of intermediate results, thus resulting in unexpected performance.

Learning from the application studies, we explore the design space for supporting data-intensive and compute-intensive applications on data-center-scale computer systems. We find two directions are promising: (i) enhancing a job control system with a set of desirable features; (ii) supporting flexible composable components and including more optimization supports in a Map-Reduce system. We plan to investigate these directions in future work.

7. REFERENCES

[1] V. Akcelik, J. Bielak, G. Biros, I. Epanomeritakis, A. Fernandez, O. Ghattas, E. J. Kim, J. Lopez, D. O'Hallaron, T. Tu, and J. Urbanic. High resolution forward and inverse earthquake modeling on terasacale computers. In *SuperComputing*, 2003.

[2] J. Bennett and S. Lanning. The netflix prize. In *KDD Cup and Workshop*, 2007.

[3] P. Buneman and W.-C. Tan. Provanance in databases. In *SIGMOD (Tutorial Track)*, 2007.

[4] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *NIPS*, 2006.

[5] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[6] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, 2007.

[7] D. DeWitt. Mapreduce: A major step backwards. http://www.databasecolumn.com/2008/01/mapreduce-a-major-step-back.html.

[8] P. Dubey. Recognition, mining and synthesis moves computers to the era of tera. *Technology@Intel Magazine*, Feb. 2005.

[9] R. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 1974.

[10] G. Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2), 1993.

[11] Hadoop. http://hadoop.apache.org/core/.

[12] J. Hays and et al. IM2GPS: Finding likely geographic locations from an image. In *CVPR*, 2008.

[13] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys*, 2007.

[14] Netflix Prize. http://www.netflixprize.com/.

[15] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: A not-so-foreign language for data processing. In *SIGMOD*, 2008.

[16] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw Hill, 2003.

[17] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*. MIT Press, 1998.

[18] H.-C. Yang, A. Dasdan, R.-L. Hsiao, and D. S. P. Jr. Map-reduce-merge: simplified relational data processing on large clusters. In *SIGMOD*, 2007.