**LITERATURE SURVEY:**

**MapReduce: Simplified Data Processing on Large Clusters**

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 easily utilize the resources of a large distributed system. Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable:  
a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google’s clusters  
every day.

The MapReduce programming model has been successfully used at Google for many different purposes. We attribute this success to several reasons. First, the model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the  
details of parallelization, fault-tolerance, locality optimization, and load balancing. Second, a large variety of problems are easily expressible as MapReduce computations. For example, MapReduce is used for the generation of data for Google’s production web search service, for sorting, for data mining, for machine learning, and many other systems. Third, we have developed an implementation of MapReduce that scales to large clusters of machines comprising thousands of machines. The implementation makes efficient use of these machine resources and therefore is suitable for use on many of the large computational problems encountered at Google.  
We have learned several things from this work. First, restricting the programming model makes it easy to parallelize and distribute computations and to make such computations fault-tolerant. Second, network bandwidth is a scarce resource. A number of optimizations in our  
system are therefore targeted at reducing the amount of data sent across the network: the locality optimization allows us to read data from local disks, and writing a single copy of the intermediate data to local disk saves network bandwidth. Third, redundant execution can be used to reduce the impact of slow machines, and to handle machine failures and data loss.

**Map Task Scheduling in MapReduce with Data Locality: Throughput and Heavy-Traffic Optimality**

Scheduling map tasks to improve data locality is crucial to the performance of MapReduce. Many works have been devoted to increasing data locality for better efficiency. However, to the best of our knowledge, fundamental limits of MapReduce computing clusters with data locality, including the capacity region and theoretical bounds on the delay performance, have not been studied. In this paper, we address these problems from a stochastic network perspective. Our focus is to strike the right balance between data-locality and load-balancing to simultaneously maximize throughput and minimize delay. We present a new queuing architecture and propose a map task scheduling algorithm constituted by the Join the Shortest Queue policy together with the MaxWeight policy. We identify an outer bound on the capacity region, and then prove that the proposed algorithm stabilizes any arrival rate vector strictly within this outer bound. It shows that the algorithm is throughput optimal and the outer bound coincides with the actual capacity

Region. Further, we study the number of backlogged tasks under the proposed algorithm, which is directly related to the delay performance based on Little’s law. We prove that the proposed algorithm is heavy-traffic optimal, i.e., it asymptotically minimizes the number of backlogged tasks as the arrival rate vector approaches the boundary of the capacity region. Therefore, the

proposed algorithm is also delay optimal in the heavy-traffic regime.

We considered map scheduling algorithms in MapReduce with data locality. We first presented the capacity region of a MapReduce computing cluster with data locality and then we proved the throughput optimality. Beyond throughput, we showed that the proposed algorithm asymptotically minimizes the number of backlogged tasks as the arrival rate vector approaches the boundary of the capacity region, i.e., it is heavy-traffic optimal.

**Joint Scheduling of MapReduce Jobs with Servers: Performance Bounds and Experiments**

MapReduce has achieved tremendous success for large-scale data processing in data centers. A key feature distinguishing MapReduce from previous parallel models is that it interleaves parallel and sequential computation. Past schemes, and especially their theoretical bounds, on general parallel models are therefore, unlikely to be applied to MapReduce directly. There are many recent studies on MapReduce job and task scheduling. These studies assume that the servers are assigned in advance. in current data centers, multiple MapReduce jobs of different

Importance levels run together. In this paper, we investigate a schedule problem for MapReduce taking server assignment in to consideration as well. We formulate a MapReduce server-job organizer problem (MSJO) and show that it is NP-complete. We develop a 3-approximation algorithm and a fast heuristic. We evaluate our algorithms through both simulations and experiments on Amazon EC2 with an implementation in Hadoop. The results confirm the advantage of our algorithms

In this paper, we studied MapReduce job scheduling with consideration of server assignment. We showed that with- out such joint consideration, there can be great performance loss. We formulated a MapReduce server-job organizer problem. This problem is NP-complete and we developed a 3- approximation algorithm MarS. We evaluated our algorithm through extensive simulation. The results show that MarS can outperform state-of-the-art strategies by as much as 40 % in terms of total weighted job completion time. We also implement a prototype of MarS in Hadoop and test it with experiment on Amazon EC2. The experiment results confirm the advantage of our algorithm

**The Hadoop Distributed File System**

The Hadoop Distributed File System (HDFS) is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. In a large cluster, thousands of servers both host directly attached storage and execute user application tasks. By distributing storage and computation across many servers, the resource can grow with demand while remaining economical at every size. We describe the architecture of HDFS and report on experience using HDFS to manage 25 petabytes of enterprise data at Yahoo.

This section presents some of the future work that the Hadoop team at Yahoo is considering; Hadoop being an open source project implies that new features and changes are d

e-ided by the Hadoop development community at large. The Hadoop cluster is effectively unavailable when its Name Node is down. Given that Hadoop is used primarily as a batch system, restarting the Name Node has been a satisfactory recovery means. However, we have taken steps towards auto-mated failover. Currently a Backup Node receives all transactions from the primary Name Node. This will allow a failover to a warm or even a hot BackupNode if we send block reports to both the primary NameNode and BackupNode. A few Hadoop users outside Yahoo! have experimented with manual failover. Our plan is to use Zookeeper, Yahoo’s distributed consensus technology to build an automated failover solution. Scalability of the NameNode has been akey struggle. Because the NameNode keeps all the namespace and block locations in memory, the size of the NameNode heap has limited the number of files and also the number of blocks address-able.

**Map-Reduce Meets Wider Varieties of Applications**

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 im-prove 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.

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 compos able components and including more optimization supports in Map-Reduce

System. We plan to investigate these directions in future work.

**Declarative Systems for Large-Scale Machine Learning**

In this article, we make the case for a declarative foundation for data-intensive machine learning systems. Instead of creating a new system for each specific flavor of machine learning task, or hard-coding new optimizations, we argue for the use of recursive queries to program a variety of machine learning algorithms. By taking this approach, database query optimization techniques can be utilized to identify effective execution plans, and the resulting runtime plans can be executed on a single unified data-parallel query processing engine.

The growing demand for machine learning is pushing both industry and academia to design new types of highly scalable iterative computing systems. Examples include Mahout, Pregel, Spark, Twister, Hadoop, and PrItr. However, today’s specialized machine learning platforms all tend to mix logical representations and physical implementations. As a result, today’s platforms 1) require their developers to rebuild critical components and to hardcode optimization strategies and 2) limit themselves to specific runtime implementations that usually only (naturally) fit a limited subset of the potential machine learning workloads. This leads to the current state of

Practice: implementing new scalable machine learning algorithms is very labor-intensive and the overall data processing pipeline involves multiple disparate tools hooked together with file- and workflow-based glue. In contrast, we have advocated a declarative foundation on which specialized machine learning workflows can be easily constructed and readily tuned. We have verified our approach with Datalog implementations of two popular programming models from the machine learning domain: Iterative Map-Reduce-Update, for deriving linear models, and Pregel, for graphical algorithms (see [5]). The resulting Datalog programs are compact, tunable to a specific task (e.g., Batch Gradient Descent and PageRank), and translated to optimized physical plans. Our experimental results show that on a large real-world dataset and machine cluster, our optimized plans are very competitive with other systems that target the given class of ML tasks (see [5]). Furthermore, we demonstrated that our approach can offer a plan tailored to a given target task and data for a given machine resource allocation. In contrast, in our large experiments, Spark failed due to main-memory limitations and Hadoop succeeded but ran an order-of-magnitude less efficiently. The work reported here is just a first step. We are currently developing the ScalOps query processing components required to automate the remaining translation steps; these include the Planner/Optimizer as well as a more general algebraic foundation based on extending the Algebricks query algebra and rewrite rule framework of ASTERIX [4]. We also plan to investigate support for a wider range of machine learning tasks and for a more asynchronous, GraphLab-inspired programming model for encoding graphical algorithms.

**Distributed Machine Learning and Graph Processing with Sparse Matrices**

It is cumbersome to write machine learning and graph algorithms in data-parallel models such as MapReduce and Dryad. We observe that these algorithms are based on matrix computations and, hence, are inefficient to implement with the restrictive programming and communication interface of such frameworks. In this paper we show that array-based languages such as R [2] are suitable for implementing complex algorithms and can outperform current data parallel solutions. Since R is single threaded and does not scale to large datasets, we have built Pronto, a distributed system that extends R and addresses many of its limitations. Pronto efficiently shares sparse structured data can leverage multi-cores, and dynamically partitions data to mitigate load imbalance. Our results show the promise of this approach: many important machine learning and graph algorithms can be expressed in a single framework and are substantially faster than those in Hadoop and Spark. Pronto advocates the use of sparse matrix operations to simplify the implementation of machine learning and graph algorithms in a cluster. Pronto uses distributed arrays for structured processing, efficiently uses multi-cores, and dynamically partitions data to reduce load imbalance. Our experience shows that Pronto is a flexible computation model that can be used to implement a variety of complex algorithms

**Using Bioinformatics Applications on the Cloud**

Dealing with large genomic data on a limited computing resource has been an inevitable challenge in life science. Bioinformatics applications have required high performance computation capabilities for next-generation sequencing (NGS) data and the human genome sequencing data with single nucleotide polymorphisms (SNPs). From 2008, Cloud computing platforms have been widely adopted to deal with the large data sets with parallel processing tools. MapReduce parallel programming framework is dominantly used due to its fast and ancient performance for data processing on cloud clusters. This study introduces various research projects regarding to reducing a data analysis time and improving usability with their approaches. Hadoop implementations and work ow toolkits are focused on address parallel

data processing tools and easy-to-use environments

These days, individual research laboratory is able to generate terabytes of data (or even larger), which is no suprises to new sequencing technologies in genomic research. High performance computation environments keep improving on processing large-scale data at low cost. The combination of MapReduce and cloud computing facilitates fast and efficient parallel processing on the virtual environment for terabyte-scale data analysis in bioinformatics, if the analysis consists of embarrassingly parallel problems. MapReduce framework is suitable for the simple and dividable tasks such as read alignment, sequence search and image recognition. Easy-to-use methods and user-friendly cloud platforms have been provided to researchers so that they can easily have ac- cess to the cloud with their large data sets uploaded on the cloud in a secure manner. Scientic work ow may focus on improving data transfer and handling tasks regarding these usability problems. More challenges are expected to deal with data storage and analysis since it grows at unprecen-dented scales.

**Comparison of Distributed Data-Parallelization Patterns for Big Data Analysis: A Bioinformatics Case Study**

As a distributed data-parallelization (DDP) pattern, MapReduce has been adopted by many new big data analysis tools to achieve good scalability and performance in Cluster or Cloud environments. This paper explores how two binary DDP patterns, i.e., Co Group and Match, could also be used in these tools. We re-implemented an existing bioinformatics tool, called Cloudburst, with three different DDP pattern combinations. We identify two factors, namely, input data balancing and value sparseness, which could greatly affect the performances using different DDP patterns. Our experiments show: (i) a simple DDP pattern switch could speed up performance by almost two times; (ii) the identified factors can explain the differences wen the “big data” era, it is very popular and effective to use DDP patterns in order to achieve scalability and parallelization. These DDP patterns also bring challenges on which pattern or pattern combination is the best for a certain tool. This paper demonstrates different DDP patterns coul

d have a great impact on the performances of the same tool. We find that although MapReduce

can be used for wider range of applications with either one or two input datasets, it is not always the best choice in terms of application complexity and performance. To understand the differences, we identified two affecting factors, namely input data balancing and value sparseness, on their performance differences. The feasibility of these two factors is verified through experiments. We believe many tools in bioinformatics and other domains have a similar logic with CloudBurst as they need to match two input datasets, and therefore could also benefit from our findings. For future work, we plan to investigate more tools that are suitable for multiple DDP patterns and their performances on other DDP engines like Hadoop, which will generalize our findings. We will also study how to utilize the identified factors to automatically

Select the best DDP pattern combination from multiple available ones.

**Compressed Nonnegative Matrix Factorization Is Fast and Accurate**

Nonnegative matrix factorization (NMF) has an established reputation as a useful data analysis technique in numerous applications. However, its usage in practical situations is undergoing challenges in recent years. The fundamental factor to this is the increasingly growing size of the datasets available and needed in the information sciences. To address this, in this work we propose to use structured random compression, that is, random projections that exploit the data structure, for two NMF variants: classical and separable. In separable NMF (SNMF) the

Left factors are a subset of the columns of the input matrix. We present suitable formulations for each problem, dealing with different representative algorithms within each one. We show that the resulting compressed techniques are faster than their uncompressed variants, vastly reduce memory demands, and do not encompass any significant deterioration in performance. The proposed structured random projections for SNMF allow dealing with arbitrarily shaped large matrices, beyond the standard limit of tall-and-skinny matrices, granting access to very efficient computations in this general setting. We accompany the algorithmic presentation with theoretical foundations and numerous and diverse examples, showing the suitability of the proposed approaches.

In this work we proposed to use structured random projections for NMF and SNMF. For NMF, we presented formulations for three popular techniques, namely, multiplicative updates, active set method for nonnegative least squares and ADMM. For SNMF, we presented a general technique that can be used with any algorithm. In all cases, we showed that the resulting compressed techniques are faster than their uncompressed variants and, at the same time; do not introduce significant errors in the final result. There are in the literature very efficient SNMF algorithms for tall-and-skinny matrices. Interestingly, the use of structured random projections allows computing SNMF for arbitrarily large matrices, granting access to very efficient computations in the general setting. As a byproduct, we also propose an algorithmic solution for

Computing structured random projections of extremely large matrices (i.e., matrices so large that even after compression they do not fit in main memory). This is useful as a general tool for computing many different matrix decompositions, such as the singular value decomposition, for example. We are currently investigating the problem of replacing the Frobenius norm with and

Norm in our compressed variants of NMF and SNMF. In this setting, the fast Cauchy transform is a suitable alternative to structured random projections. Compression consists of sampling and rescaling rows of A, thus identifying the so-called corset of the problem. This formulation is of particular interest for network analysis, where we need to deal with sparse structures.