Gilbert: A sparse linear algebra environment for machine learning on web-scale data.

Exposé master's thesis

Till Rohrmann
Technical University of Berlin, Germany
till.rohrmann@mailbox.tu-berlin.de

1. INTRODUCTION

Nowadays, computing devices are almost ubiquitous in our daily lives. They ceaselessly gather, process and store information of any kind and their computing and storage capacities grow every year. These capacities exhibit an almost exponential growth rate [13] like the growth of the number of components on integrated circuits [18], known as Moore's law. Nowadays, it is estimated that 2.5 ZB of data is created each day and this number is supposed to rise up to 40 ZB by the year of 2020 [3].

With the increase of data, people became aware of its usefulness and thus the demand for analyzing tools, able to exploit the gathered data, increased as well. Currently, insights gained from collected data already helps to improve a variety of processes. For example, retail stores analyze their sales, customer, pricing and weather data in order to decide which products to offer or when to do a discount sale. Police departments try to detect probable crime locations by extracting patterns from previously recorded criminal acts and then reinforce the policemen in this region [15]. Hospitals analyze their patients' records and scientific studies in order to find the cancer treatment best complying with the specifics of the patient and thus having the highest chance of cure [2]. These examples underline the importance and utility of data analysis tools. The majority of these tools are based on statistic techniques. Consequently, they can improve their descriptive and predictive results by getting fed more data. However, this comes at the price of an increased computing time which requires faster computer systems to make the computation feasible.

Clock rates of CPUs stagnate, because technology hit the power wall. In order to further increase computing power though, multi-core and distributed systems were built. However, these systems pose new challenges for programmers, since now they have to know about locking, deadlocks, race-conditions and inter-process communication in order to make most of the available hardware. Due to this, parallel program development became cumbersome and error-prone. Therefore, new programming models were conceived which relieve the programmer from the tedious low-level tasks related to parallelization such as load-balancing, scheduling of parallel tasks and fault recovery.

These are the reasons why Google's MapReduce [9] framework and its open source reimplementation Hadoop [1] became so popular among scientists as well as engineers. MapReduce is a programming framework for concurrent computations on vast amounts of data running on a large cluster. Due to its simplicity, it is widely adopted and countless distributed applications exist using this framework.

MapReduce, however, exhibits also some deficiencies in realizing common sub-tasks such as joining two datasets. These deficiencies result from the strict programming model and cause some serious performance losses. There are some other distributed programming models such as Stratosphere [6] and Spark [21] which try to tackle these problems. They both support iterative algorithms which is a necessary requisite for many data analytical algorithms. Furthermore, they both offer a more flexible pipeline not requiring to store intermediate results always to hard disks.

But still, these frameworks force the user to express the program in a certain way, which is often not natural or intuitive for a user coming from a different domain. This implies that one has to overcome a particular entry-barrier to use the system and this might already be too high for some users. Furthermore, the actual program might become lengthy and complicated expressed within the programming model. This makes developing and debugging the program difficult as well as time-consuming and thus expensive. Especially in the field of data analytics and machine learning programs are usually expressed in a mathematical form. Therefore, systems such as Matlab and R are widely used and recognized for their fast prototyping capabilities and their extensive mathematical libraries. However, these linear algebra systems lack proper support for automatic parallelization on large clusters and thus restricting the user to a single workstation. Therefore, the amount of processable data is limited to the size of the main memory, which constitutes a serious drawback for real-world applications. Moreover, machine learning people are usually no experts in the field of distributed computing, and neither vice versa. The group of experts on both fields is negligible small. Consequently, it is very difficult, time-consuming and not least expensive to implement machine learning algorithms on distributed systems.

This problem would be mitigated by having a distributed sparse linear algebra system supporting a Matlab- and

TU Berlin, Database Systems and Information Management, 2014, Germany
.

R-like language. Assuming that such a system is realizable, then one could run existing Matlab- or R-code directly or at most with minor adjustments in a distributed fashion. Furthermore, new distributed algorithms could be quickly implemented benefiting from the expressiveness of linear algebra. This would drastically speed up the application of machine learning and data analysis algorithms on web-scale data. Therefore, I want to research and implement a distributed sparse linear algebra system, henceforth called Gilbert, in the context of my master's thesis.

2. THESIS APPROACH

Within my master's thesis I want to answer the following questions:

- Which are the language primitives of a distributed sparse linear algebra system in order to support a multitude of current machine learning algorithms?
- How can the supported operations be realized in a distributed system?

2.1 Language primitives

I intend to implement an intermediate representation of a Matlab program. The additional abstraction layer allows language independent optimizations and makes the system independent from the actually used frontend language. The set of language primitives of the intermediate representation has to include the operational primitives of linear algebra as well as an iteration abstraction in order to realize algorithms based on convergence. Furthermore, it is of particular interest to keep this set as small as possible, because this would alleviate a possible optimization step prior to execution.

2.2 Runtime

I want to research how the most common linear algebra operations can be efficiently represented within a parallel data flow system. One key aspect is going to be the matrix matrix multiplication because it is one of the central operations of many machine learning algorithms and it is one of the most time consuming operations as well. Therefore, it is crucial to efficiently implement this operation. This requires a thorough investigation and evaluation of all possible implementations. Since this will strongly depend on the actual distributed system, I will implement and evaluate these operations within Stratosphere. The Stratosphere implementation acts at the same time as a proof of concept of Gilbert. I have chosen Stratosphere over other systems, such as Spark, because of its expressiveness and its powerful parallel data-flow optimizer. Last but not least, Stratosphere is developed at the chair of Prof. Markl, where I write my master's thesis. However, there should not be any fundamental obstacle which prevent the utilization of an other distributed computing system.

2.3 Optimization

If I have still some time left, then I will research possible optimization strategies for linear algebra programs. One obviously important choice of execution would be the order of subsequent matrix matrix multiplications. Different execution plans might emit different intermediate results which vary in size and density and thus making one of the plans favorable over the other. I intend to use a similar cost-based plan enumeration approach as it is used for database query optimizations.

2.4 Implementation

The distributed sparse linear algebra system shall have a Matlab- or R-like language frontend. For the sake of simplicity, I will implement a subset of the Matlab language. In order to support this subset, a Matlab parser has to be implemented. Additionally to the parser, I will also implement a static typing mechanism so that possible errors resulting from incompatible types are already caught at compile time. I will use Scala for all implementations tasks, because it nicely integrates in the Java ecosystem and at the same time enriches it with valuable additions like functional programming support and parser combinators.

2.5 Evaluation

I want to demonstrate the proper functioning and potential of Gilbert by comparing the runtime of different machine learning algorithms implemented in this system with those implemented in a traditional MapReduce framework. Hopefully, the results of the automatically parallelized algorithms are comparable to the hand-tuned versions. Furthermore, I like to show the applicability of the proposed system to web-scale analytical processing. This will require a thorough examination of its scalability behavior. As benchmarking algorithms I intend to implement the PageRank [19], non-negative matrix factorization [20] and k-means [16] algorithm. These algorithms entail to support the basic linear algebra operations as well as an iteration mechanism. Furthermore, they are examples of widely used algorithms and thus they emphasize the relevance of the developed system.

3. SCHEDULE

My schedule for the remaining things to do looks the following: First, I will finish the implementation of Gilbert. At the moment, the system can parse and compile Matlab programs using the specified subset of the language. However, it cannot be executed on Stratosphere yet. I intend to implement the Stratosphere support in the next 1 to 1.5 months.

After this is done, I will start with the evaluation of the system. For this purpose, I will implement the benchmark algorithms and investigate their behavior with respect to varying input and cluster sizes. Since I assume that some unrecognized problems will surface in the context of this evaluation, I plan 1 month for the evaluation tasks. In the case that there are some grievous errors in the implementation, I schedule another 0.5 month for reworking purposes.

Last but not least, I will finish my thesis by compiling the findings of my work. Since thesis writing is an iterative process I calculate 1.5 months for it to be done. In order to minimize the work load at the end, I try to start writing after I have finished my implementation.

4. RELATED WORK

The challenges entailed by harnessing vast amounts of data propelled much research on the field of distributed machine learning to subdue the data flood. Consequently, several frameworks and programming models have been proposed to tackle the aforementioned problems. One inspiring approach is SystemML [11]. SystemML provides a declarative higher-level language for expressing linear algebra operations. These operations are then mapped onto MapReduce

tasks. It further applies several optimization strategies such as blocking, dynamic block-level operations, piggybacking to reduce the number of MapReduce tasks and local aggregation within the reducers. The system can evaluate different execution plans depending on the size of its inputs and its blocking strategy. It selects the most cost-efficient plan based on the network costs. SystemML is similar to the proposed approach, however it has no proper iteration mechanism besides of simple loop unrolling. Furthermore, it relies on the MapReduce framework with all of its deficiencies compared to the more powerful distributed computing system Stratosphere. The declarative language of SystemML is not a subset of Matlab or R and thus it cannot directly be used by native Matlab or R users.

Stratosphere [6] is a distributed computing framework which employs PACTs [4], a generalization of MapReduce. It adds additional 2^{nd} level functions, which are called input contracts in the context of Stratosphere, such as *match*, *cross* and *coGroup* in order to improve the expressiveness and efficiency of the MapReduce paradigm. Furthermore, it adds the concept of output contracts which annotate input contracts with certain properties. These properties are exploited by the compiler to select the best execution plan. Recently, the framework has been extended to support bulk and incremental iterations [10]. However, it still lacks an easy to use language for the development of machine learning algorithms.

Another distributed computing framework is Spark [21] which can be seen as an extension of MapReduce with iteration support. Spark performs its computation on resilient distributed datasets (RDD) which can be kept in memory during computations. This allows to efficiently realize iterations within in the framework. Spark lacks an intuitive Matlab-like language frontend and thus it requires a considerable expertise to code well performing algorithms.

Apache Mahout [5] is a project offering a library of scalable machine learning algorithms. Many algorithms use the MapReduce paradigm to achieve scalability and are written in Hadoop. However, Mahout offers no easy way to write new algorithms by using a declarative language for example. By using Hadoop, the actual execution plan of an algorithm has to be hand-tuned for the specific cluster and input size.

Pegasus [14] is a programming model mainly intended for graph mining purposes. It is centered around the abstraction of a generalized iterative matrix vector multiplication (GIM-V), which can be found in many graph algorithms. The GIM-V operation can be represented by a map and reduce operation. Consequently, Pegasus implements it on top of Hadoop. Gilbert is supposed to contain the GIM-V operation and will not only consider the operation itself for optimization purposes but also its context of previous and succeeding operations. Therefore, Gilbert should be a generalization of Pegasus.

Y. Bu et al. [8] remarked that there exist a multitude of more or less specialized programming models for the task of distributed machine learning. All of these systems exhibit a tight coupling of a solution's logical representation and physical representation. This renders optimization difficult, because one is bound to the underlying runtime implementation and disregards alternative execution strategies. Moreover, the systems are mostly disjoint which implies that each framework has to be updated in order to profit from new optimization strategies. As a solution the authors propose to employ Datalog as a declarative language for the specification of higher level programming models such as iterative MapReduce or Pregel [17] within their system. Standard query optimization techniques are applied on this common intermediate representation and then it is transformed into a physical execution plan. This plan is executed by Hyracks [7], a data-parallel platform for data-intensive tasks. This approach has the advantage that a wider class of machine learning algorithms are efficiently supported within the same system, thus profiting from the same underlying infrastructure. However, at the moment the physical plans are still created by hand which makes the framework not applicable yet.

L. Hendren [12] developed a static typing system for Matlab. Types are defined by a special keyword so that the system needs a weaver to convert the typing statements into valid Matlab code. The defined types can then be used by the compiler to generate more efficient code and to check runtime types against their specifications. Furthermore this contributes to a better documentation of code and makes understanding easier.

5. REFERENCES

[1] Hadoop. `http://hadoop.apache.org/`, 2008. Cited January 8th 2014.

[2] Ibm watson hard at work: New breakthroughs transform quality care for patients. `http://www.mskcc.org/pressroom/press/ibm-watson-hard-work-new-breakthroughs-transform-quality-care-patients`, 2013. Cited January 8th 2014.

[3] Big data within ibm. `http://www.ibm.com/big-data/`, 2014. Cited January 8th 2014.

[4] A. Alexandrov, S. Ewen, M. Heimel, F. Hueske, O. Kao, V. Markl, E. Nijkamp, and D. Warneke. Mapreduce and pact-comparing data parallel programming models. In *BTW*, pages 25–44, 2011.

[5] Apache. Apache mahout. `http://mahout.apache.org/`, 2011. Cited January 9th 2014.

[6] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke. Nephele/PACTs: a programming model and execution framework for web-scale analytical processing. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 119–130. ACM, 2010.

[7] V. Borkar, M. Carey, R. Grover, N. Onose, and R. Vernica. Hyracks: A flexible and extensible foundation for data-intensive computing. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1151–1162. IEEE, 2011.

[8] Y. Bu, V. Borkar, M. J. Carey, J. Rosen, N. Polyzotis, T. Condie, M. Weimer, and R. Ramakrishnan. Scaling datalog for machine learning on big data. *arXiv preprint arXiv:1203.0160*, 2012.

[9] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[10] S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl. Spinning fast iterative data flows. *Proceedings of the VLDB Endowment*, 5(11):1268–1279, 2012.

[11] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan. SystemML: Declarative

machine learning on MapReduce. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 231–242. IEEE, 2011.

[12] L. Hendren. Typing aspects for matlab. In *Proceedings of the sixth annual workshop on Domain-specific aspect languages*, pages 13–18. ACM, 2011.

[13] M. Hilbert and P. López. The worlds technological capacity to store, communicate, and compute information. *Science*, 332(6025):60–65, 2011.

[14] U. Kang, C. E. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 229–238. IEEE, 2009.

[15] S. Lohr. The age of big data. *New York Times*, 11, 2012.

[16] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14. California, USA, 1967.

[17] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[18] G. E. Moore et al. Cramming more components onto integrated circuits, 1965.

[19] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[20] D. Seung and L. Lee. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13:556–562, 2001.

[21] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.