cnidaria: A Generative Communication Approach to Scalable, Distributed Learning

Michael J. Kane
Yale University
New Haven, CT 06511
michael.kane@yale.edu

Bryan W. Lewis
Paradigm4
Waltham, MA 02451
blewis@illposed.net

Abstract

This paper presents a scalable, software framework that facilitates large-scale learning and numerical computing. Unlike existing MapReduce frameworks our design is not limited to embarrassingly parallel computing challenges. The framework sits on top of existing storage infrastructures and results of a computation may left out on the cluster (a reduce step is not required). Unlike existing distributed numerical frameworks the proposed framework is elastic and works with both dense and sparse data representations. This generality is achieved through a generative communication scheme whose expressions are either consumed by the distributed computing environment or used to move data, in a peer-to-peer (P2P) fashion, between nodes in a cluster/cloud. This approach integrates advances in the both cloud computing and the distributed numerical computing community and can be applied to a general class of learning challenges.

1 Categorizing computational learning challenges

Efficient learning of very large data sets is essentially a matter of managing the computational complexity. The complexity is always a function of the size of the data, and as a result, management falls into one of three strategies: subsetting; divide and recombine; or reduce the analysis.

Subsetting The first strategy, subsetting, applies a learner to a smaller version of the training data. For learners that can achieve a high degree of accuracy with reasonable size data sets this is an effective strategy. Data that are not included in learning may represent only a small increase in accuracy at a disproportionally high cost in computational complexity.

While sampling is effective when the distribution of the data is relatively simple and can be encoded as a small subset of the data, this is not always the case. There are challenges where all of the data need to be used so that the model can encapsulate all of their nuance and diversity.

Divide and recombine To both incorporate large data into a training set and manage computational complexity researchers have often taken the second approach of dividing the data into partitions and creating learners for each partition. An ensemble prediction is then created by allowing each learner to predict on the partitioned data and then averaging over those predictions.

The complexity justification for this approach was presented in Matloff (2013) and is based on the observation that computational complexity is a function of data size. Let n be the number of samples in the data set and assume that the cost to train a learner is $O(n^\alpha)$ where $\alpha \geq 2$. Let c be the number of parallel processes that can be used to train the learner. Then the best computation complexity we can achieve for training a single learner is $O(n^\alpha/c)$. Assume instead that we partition the samples in the data so that each partition has size n/c. In this case we can train each learner on a partition in parallel. The cost of training each learner is reduced to $O((n/c)^\alpha)$ and the total complexity is therefore $O(c(n/c)^\alpha)$ or $O((n/c)^\alpha)$ when each learner is trained on one of c parallel processes.

Statistical justifications have been given showing that the ensemble learners converge asymptotically to the equivalent learner trained on the entire data set, the reference learner, in numerous settings for a large class of learners (Kleiner *et al.*, 2011; Guha *et al.*, 2012; Matloff, 2013). However, all of these justifications rest on the assumption that the data are i.i.d. and that the data are distributed similarly across partitions. The justification works by showing that each learner is capable of providing a noisy estimate for predictions and then showing that the average of these noisy estimates converges in distribution to the reference.

Although the theoretical and methodological framework is well-developed, partitioning in this second category poses a practical computational challenge. Namely, for the learners to converge, the partitions must be similarly distributed. Kleiner *et al.* (2011) address this by assuming a random partitioning of the data. For implementation, this issue is not so easily addressed. Distributed data often have a natural partitioning over a cluster, and the data partitioning is often influenced by when and where they are collected. As a result, it is not uncommon for a sample to be close to other similar samples and finding a practical solution to the problem of assuring similarly distributed partitions is difficult in general.

Analysis reduction The divide and recombine approach is a powerful technique for incorporating all data from a large data set into a learner and it can be applied to a large class of learning challenges. However, it does not address all learning challenges. In particular, there is a large class of extremal value challenges in learning that do not lend themselves to the second approach. The singular value decomposition provides a canonical example. For these types of problems, the algorithms themselves must be reexamined and reimagined so that only salient information for an analysis can still be derived. Baglama and Reichel (2006) and Liberty *et al.* (2007) epitomizes this methodology, providing singular value information for only the largest or smallest singular values and vectors through an iterative approach thereby allowing the user to select the relevant information (number of singular values/vectors) and control the accuracy by specifying a convergence accuracy. These truncated singular value decomposition (TSVD) techniques facilitate traditional applications, like recommender systems, on data that are orders of magnitude too large to apply to their conventional analogues.

This third category is generally the most difficult to manage at scale with modern software technologies because of the type of calculations that are required. In the case of the TSVD, the difficult computation is the matrix multiply. The distributed block matrix multiply is an outer-product of inner-products of row-blocks from one matrix and column blocks of another. Each block matrix multiply requires that both matrices involved are on the same node and, as a result, it requires a sophisticated communication scheme, well beyond what can be achieved efficiently with a Map and Reduce (Kazmi *et al.*, 2013).

2 Framework overview

Until now, there has not been a unified approach to addressing the communication needs of all three described categories that is also elastic, making use of new computing resources as they are available and compensating for hardware faults as they occur. This paper describes an elastic communication framework for the general problem of learning on distributed data. The framework's design is general but it requires two key pieces of software infrastructure. First is a distributed configuration service that can broadcast to all processes in the cluster and route tasks to specific processes. Second is a P2P communication mechanism that allows processes to communicate directly with each other, without needing to go through the configuration service.

These infrastructure systems provide communication among distributed processes for two very different purposes. The configuration service manages the location of computing resources, the location of distributed data, and tasks that are consumed by the cluster. These are responsibilities related to the state and coordination of the distributed system. This is inherently a centralized system providing cluster services across the cluster. The P2P mechanism moves data between processes in the distributed system, using the configuration service to broker connections across the cluster. While the connection brokering is a relatively inexpensive operation, data movement is not. With a separate P2P mechanism the majority of data movement is guaranteed not go through the configuration service, making maximal use of communication resources.

As mentioned previously, the configuration service manages computing and data resources. Computing resources refer to processes or collections of worker processes available to perform distributed tasks. Data resources refer to blocks of data that may be used in a distributed computation. These blocks are managed by workers "listening" for tasks that involve the resources they manage. Computations can therefore be localize to either specified workers or data thereby minimizing data movement. When a worker is executing a task that requires another resource that it does not manage, then it uses the configuration service to broker a P2P connection so that the required resource can be received.

In this framework all communication is mediated by the configuration manager. Processes do not keep track of other processes as with the classic master/slave model employed by MPI (Snir *et al.*, 1995). Instead, we propose a *decentralized* model where any distributed process in the cluster can initiate a computation to be consumed by the rest of the distributed processes. Furthermore, the parallel processes in this architecture are *anonymous*. With the exception of a P2P transfer, processes are not aware of other distributed processes. Computations are defined in terms of the resources being computed on.

The combination decentralization and anonymity of processes in this framework provide elasticity and performance through redundancy. Data resources can be served by multiple processes in the cluster so that should a process fail, the task that was submitted to the configuration manager will timeout and the task is resubmitted. Fault-tolerance is achieved since, as long as the data resource is available elsewhere on the cluster, a task that requires it will be served. Likewise, a computation can easily be "scaled-out" by adding new processes to the cluster. In this case, a new process announces its availability to the configuration manager. The new process can be used immediately by the cluster, increasing the speed of execution.

The ability to add new computing resources as a computation is being performed can also be used to speed-up data intensive operations. In this setting, a computation is "slow" because a data resource cannot be served quickly enough to keep up with the inflow of tasks that depend on it. This is easily remedied by replicating the heavily-used resource, either in a computing resource that is not being used or with a new computing resource. Therefore the performance of data intensive calculations, like those associated with many learning tasks can be managed through the availability of computing and data resources in the cluster.

The proposed framework is designed to be data-store agnostic. That is, as long as data can be served, whether it is from a database, file, etc., it can be utilized in a computation. However, the framework is still subject to the CAP theorem (Gilbert and Lynch, 2002), meaning a communication framework of this kind can provide at most two of the following: consistency (data resources are the same same at all times), availability (every tasks receives a response, even if it fails), and partition tolerance (tasks are served, even when there are failures in the system). The framework's meta-data consistency properties are determined by the configuration service technology. The data consistency model is currently under investigation with the goal of providing eventual consistency (Vogels, 2009).

3 Generative communication for machine learning

The proposed design supports the use of and sophisticated communication schemes, but the cluster grammar is composed of only two functions, each requiring three parameters. First is the `pull` function, which takes as a first argument the computing or data resource where the calculation should be executed. The second argument is an expression that will be evaluated at the resource location. The third argument is a boolean flag indicating if the task should go to all resources with the handle specified by the first argument or first specified resource able to consume the task. The expression is routed to the resource, evaluated, and the result is returned to the process initiating the call. Second is the `push` function, which takes the same two arguments as `pull`. The difference is rather than returning the evaluated expression, the result is left on the evaluating process and served as a new data resource. The handle to the newly created data resource is returned to the initiating process. A `push`/`pull` call is valid from any process in the distributed environment and guaranteed to produce the same result.

This communication scheme is *generative* as defined by Gelernter (1985) and satisfies two key criteria. The first criteria is *communication orthogonality*, which specifies that a process receiving

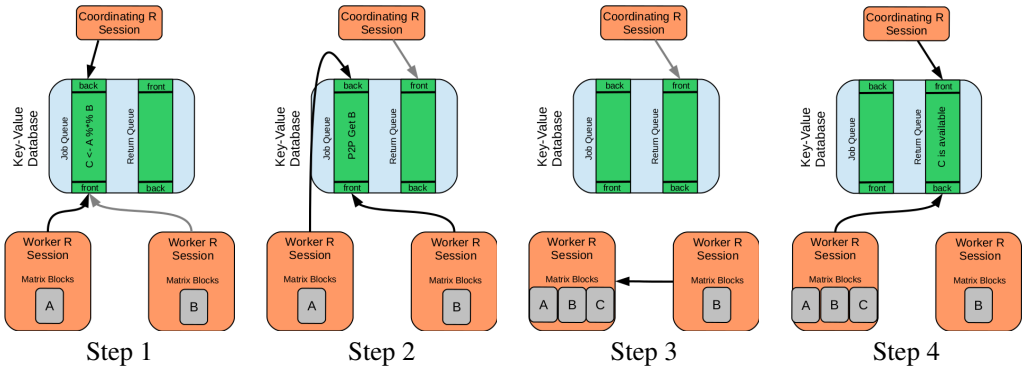

Figure 1: The steps required to perform a matrix multiply for two matrices, A and B, where the matrices reside in two different processes.

data has no prior knowledge about the corresponding sender and, at the same time, the sender has no prior knowledge about the receiver. The result is that resource handles that are served within a cluster are valid in all distributed processes in that cluster (space uncoupling). Also, a resource remains available to processes in the cluster until it is explicitly removed (time uncoupling). Space and time uncoupling result in resources are maintained automatically, a property known as distributed sharing. The second criteria is *free naming* and achieved through two properties. The first is support for continuation passing. That is, resource handles can hold other resource handles and can be stored in data structures, just as any other type. The second is structured naming, which means that resource handles may be content-addressable. This is achieved by explicitly querying and removing objects in a broadcasted `pull` expression.

4 A simple example: matrix multiply

To show more concretely how the framework performs distributed operations consider the challenge of multiplying two matrices `A` and `B` that are managed by different processes in a distributed R computing environment. The following steps describe the required operations and Figure 1 shows them graphically.

1. The coordinator presents a new task to the cluster. In this case, a matrix multiply of `A %*% B` with `C` as the result. Since the task requires both `A` and `B` either worker process can consume the task. In this case the worker with `A` will consume it.
2. Since the worker consuming the tasks does not have all of the data required to complete the calculation, it puts in a peer-to-peer request to receive `B`. This task is consumed by the worker with `B`.
3. `B` is sent and the matrix multiplication is performed resulting in matrix `C`.
4. A handle to `C` is returned to coordinator indicating that the computation has been performed and that the result is available. Note that `C` is not returned, only its availability.

5 Status and future work

The described framework is currently under heavy development and is being implemented as a set of packages for the R (R Core Team, 2013) programming environment. The package is named **cnidaria** (Kane and Lewis, 2013), after a phylum of simple, hearty aquatic animals. The implementation is modular and compatible with any number of configuration managers and P2P libraries. However, cnidaria currently relies on Redis (Sanfilippo, 2013) configuration management and ØMQ (iMatix Corporation, 2013) for P2P connections. The distributed computing capabilities are available through the low-level `pull` and `push` functions, described earlier as well as a set of high level data structures (`vector`, `matrix` (both sparse and dense), and `data.frame` with `array` planned). These data structures behave similar to the analogous native R data structures allowing users to leverage existing functionality provided by both the language and contributed packages.

Acknowledgments

The authors would like to thank AT&T Labs-Research for its support and guidance in this research and software development effort.

A portion of this research is based on research sponsored by DARPA under award FA8750-12-2-0324. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

Disclaimer

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

References

Baglama J, Reichel L (2006). "Restarted block Lanczos bidiagonalization methods." *Numerical Algorithms*, **43**(3), 251–272.

Gelernter D (1985). "Generative communication in Linda." *ACM Trans. Program. Lang. Syst.*, **7**(1), 80–112. ISSN 0164-0925. doi:10.1145/2363.2433. URL http://doi.acm.org/10.1145/2363.2433.

Gilbert S, Lynch N (2002). "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services." *ACM SIGACT News*, **33**(2), 51–59.

Guha S, Hafen R, Rounds J, Xia J, Li J, Xi B, Cleveland WS (2012). "Large complex data: divide and recombine (D&R) with RHIPE." *Stat*, **1**(1), 53–67. ISSN 2049-1573. doi:10.1002/sta4.7. URL http://dx.doi.org/10.1002/sta4.7.

iMatix Corporation (2013). *ØMQ: Distributed Computing made Simple*. URL http://zeromq.org/.

Kane MJ, Lewis B (2013). *cnidaria: a computing framework for writing distributed data structures and algorithms*. URL https://github.com/kaneplusplus/cnidaria.

Kazmi S, Kane MJ, Krauthammer M (2013). "Benchmarking Technology Infrastructures for Embarrassingly and Non-embarrassingly Parallel Problems in the Biomedical Domain." In "Proceedings of the ORNL Biomedical Sciences and Engineering Conference," .

Kleiner A, Talwalkar A, Sarkar P, Jordan MI (2011). "A scalable bootstrap for massive data." *arXiv preprint arXiv:1112.5016*.

Liberty E, Woolfe F, Martinsson PG, Rokhlin V, Tygert M (2007). "Randomized algorithms for the low-rank approximation of matrices." *Proceedings of the National Academy of Sciences*, **104**(51), 20167–20172.

Matloff N (2013). "Software Alchemy: Turning Complex Statistical Computations into Embarrassingly Parallel Ones." *Preprint*.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Sanfilippo S (2013). *Redis: Advanced Key-Value Store*. URL http://redis.io.

Snir M, Otto SW, Walker DW, Dongarra J, Huss-Lederman S (1995). *MPI: the complete reference*. MIT press.

Vogels W (2009). "Eventually consistent." *Commun. ACM*, **52**(1), 40–44. ISSN 0001-0782. doi:10.1145/1435417.1435432. URL http://doi.acm.org/10.1145/1435417.1435432.