Design Principles for Modular and Scalable Scientific Analysis Systems

ABSTRACT

Revolutions in data acquisition are drastically changing how science conducts experiments. For example, "next-generation" sequencing technologies has driven exponential growth in the volume of genomic data, and similar trends impact many fields which rely on imaging, such as astronomy and neuroscience. However, many traditional "scientific computing" systems are a poor fit for these analyses, as they either provide poor programming abstractions, or require too much effort to program.

In this paper, we introduce a set of principles for decomposing scientific analysis systems so that they can be implemented efficiently on top of existing systems, while providing productive programming interfaces. We motivate these principles with an example genomics pipeline which leverages open-source MapReduce and columnar storage techniques to achieve a $> 50\times$ speedup over traditional genomics systems, at half the cost.

Categories and Subject Descriptors

L.4.1 [**Applied Computing**]: Life and medical sciences—*Computational biology*; H.1.3.2 [**Information Systems**]: Data management systems—*Database management system engines, parallel and distributed DBMSs*; E.3.2 [**Software and its Engineering**]: Software creation and management—*Software Development Process Management*

General Terms

Design

Keywords

Analytics, MapReduce, Genomics, Scientific Computing

1. INTRODUCTION

With major improvements in scientific data acquisition techniques, data storage and processing have become major problems for scientists [16, 5]. In fields like neuroscience [7] and genomics [19], scientists routinely perform experiments that use terabytes (TB) to petabytes (PB) of data. While traditional scientific computing platforms are optimized for fast linear algebra, many emerging domains make heavy use of statistical learning techniques coupled with user defined operations on top of semistructured data. This move towards statistical techniques has been driven by the increase in the amount of data available to scientists, as well as the rise of statistical systems which are accessible to non-experts, such as `Scikit-learn` [15] and `MLI` [18].

While the increase in the amount of scientific data available is a boon for scientists, it puts significant stress on existing tool chains. Using the current "best practice" genomics software,[1] it takes approximately 120 hours to process a single, high-quality human genome using a single, beefy node [20]. Similarly, these large datasets are costly to store. To address these challenges, scientists have applied computer systems techniques such as MapReduce [14] and columnar storage [8] to custom scientific compute/storage systems.

In this paper, we argue that a system composed of the correct mix of computing systems can provide better performance and scalability than custom systems, while enhancing the abstractions exposed to scientists. By building a system using Apache Avro, Parquet, and Spark [1, 2, 24], we were able to achieve a $50\times$ increase in throughput over the current "best practice" pipeline. In the process of creating this system, we developed a "narrow waisted" layering model for building similar scientific analysis systems. This narrow waisted model is inspired by the OSI model for networked systems [25]. We demonstrate the generality of this model by using it to implement a system for processing astronomy images.

In addition to the architecture and systems we will introduce in this paper, scientific systems require specialized join and access patterns. Specifically, joins may need to be performed in a coordinate space, and the total size of a dataset may be too large to be kept on local disk. In this paper, we also introduce algorithms for performing coordinate space joins, and discuss mechanisms for improving the efficiency of loading datasets from remote block stores.

2. BACKGROUND

This section will compare and contrast the various "big data" analysis systems with existing scientific systems.

[1] BWA [11] and the Genome Analysis Toolkit [14, 6].

1. MapReduce-based workflows
   (a) In CS, development of MapReduce → Hadoop → Spark
   (b) Equivalent systems in bioinformatics → GATK [14]
   (c) Hadoop-based genomics tools [17, 10]
   (d) Use of Spark for neuroscience
2. Database driven systems
   (a) SciDB [4]
   (b) GQL [9, 3]
3. Storage layers
   (a) CRAM [8]
   (b) YT [21]

Takeaways:

- There are significant computer system design problems in science:
  1. Compression → column stores (CRAM [8], YT [21])
  2. Performance/parallelism → MapReduce (GATK [14])
- Traditional SC/DB systems provide poor abstractions for most scientists
- As a result, there are lots of "roll your own" systems in science

3. PRINCIPLES FOR SCIENTIFIC ANALYSIS SYSTEMS

3.1 Workloads

1. Characteristics of data
   (a) Scientific data tends to be sparse
   (b) Different users want to look at different subsets of both rows and columns
   (c) Data may not always be in a single site, or stored locally
   (d) *Experimental data* is immutable.
   (e) What are access patterns?
2. Characteristics of a ideal storage system:
   (a) Efficient support for projection of different columns
   (b) Efficient support for per-record predicates
   (c) Should not relegate user to a single execution environment
3. Processing:
   (a) Workloads are highly variable by field
   (b) For genomics, workloads are trivially data-parallel
   (c) Similar for fields with heavy image processing workloads
   (d) Simulation based fields are tougher; have all-to-all computation pattern, run on supercomputer
   (e) Defer discussion to §3.3
   (f) Ideally, cross-platform.

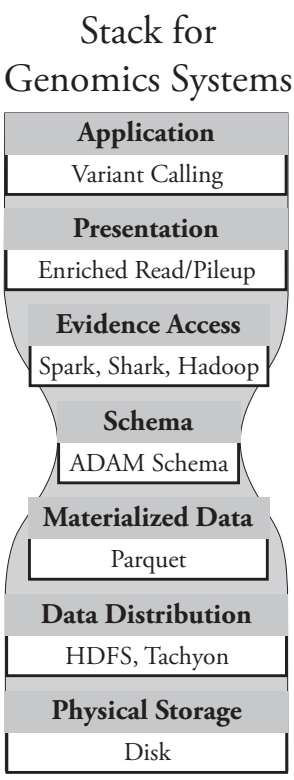

Figure 1: A Stack Model for Scientific Analysis

3.2 Layering

Discussion of Figure 1.

Specifically, we need to:

1. Show how current systems fit into the stack model, and how our proposed stack is different
2. Elucidate why it is more efficient to build systems that are decomposed as per our stack above (reference networking stack and protocol interchange), see Bafna et al [3], talk about costs of programming without good stack model

3.3 Execution Platforms

TL;DW; need to have a good discussion of what applications are good for MapReduce, what are good on top of a database, what are good on an HPC farm, what should be done with an abacus, etc. Needs to be written carefully to show the virtues of the Figure 1 stack, while being frank about weaknesses.

4. IMPLEMENTATION

4.1 Genomics Pipeline

Compare/contrast to current pipelines; talk about what the stages do in reasonable but not excessive detail. Make reference to WHAM [12] to show that this is an application domain that SIGMOD has determined to be important.

4.2 Coordinate System Joins

This will be a compare/contrast discussion of the multiple join algorithms we've created. TBD.

4.3 Loading Remote Data

1. Data may not be kept locally
   (a) Too much data to keep locally
   (b) Not all data is hot
2. May push data off local disks into block store
3. Manually re-staging data has high latency cost → impacts throughput
4. What do we need to do to accommodate this?
   (a) Efficient indexing
   (b) Remote push-down predicate
5. Discuss S3/Parquet interaction

5. PERFORMANCE

This section will address:

- Performance of ADAM on real datasets
- Compression achieved by Parquet
- Examples extending the proposed stack to Astronomy

Experiments to run:

- General demonstration of scaling for genomics pipeline; updated experiments from TR
- Experiments on coordinate system joins; broadcast vs. partition join strategies
- Experiments showing benefit from performing remote data access without staging

6. DISCUSSION

6.1 Scientific Processing on MapReduce

Big critique from SciDB camp is that MR is an inappropriate platform for scientific computing due to lack of support for linear algebra. We need to counter this point, by allusion to performance on algorithms above, and by alluding to specialized libraries for ML & graph processing [18, 23].

Also, note that we don't argue that MR is the correct platform for particle simulations and other traditional MPI workloads. However, MPI is the *wrong* platform for most analyses.

Also, note that most scientific workloads require applying a UDF across a large set of data. This is not inefficient to *run* on a database, but it is inefficient to *write*; SQL is a poor language for scientific/statistical computing.

6.2 Cost of Non-Commodity Systems

The advantage of the stack model we propose is that it enables the use and reuse of commodity systems, instead of reinventing the wheel (or, inventing a *slightly* different wheel).

7. CONCLUSION

In the end, we conclude.

APPENDIX

A. REFERENCES

[1] Apache. Avro. `http://avro.apache.org`.

[2] Apache. Parquet. `http://parquet.incubator.apache.org`.

[3] V. Bafna, A. Deutsch, A. Heiberg, C. Kozanitis, L. Ohno-Machado, and G. Varghese. Abstractions for genomics. *Communications of the ACM*, 56(1):83–93, 2013.

[4] P. G. Brown. Overview of SciDB: large scale array storage, processing and analysis. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 963–968. ACM, 2010.

[5] J. P. Cunningham. Analyzing neural data at huge scale. *Nature methods*, 11(9):911–912, 2014.

[6] M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. del Angel, M. A. Rivas, M. Hanna, et al. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature genetics*, 43(5):491–498, 2011.

[7] J. Freeman, N. Vladimirov, T. Kawashima, Y. Mu, N. J. Sofroniew, D. V. Bennett, J. Rosen, C.-T. Yang, L. L. Looger, and M. B. Ahrens. Mapping brain activity at scale with cluster computing. *Nature methods*, 11(9):941–950, 2014.

[8] M. H.-Y. Fritz, R. Leinonen, G. Cochrane, and E. Birney. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome research*, 21(5):734–740, 2011.

[9] C. Kozanitis, A. Heiberg, G. Varghese, and V. Bafna. Using Genome Query Language to uncover genetic variation. *Bioinformatics*, 30(1):1–8, 2014.

[10] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg. Searching for SNPs with cloud computing. *Genome Biology*, 10(11):R134, 2009.

[11] H. Li and R. Durbin. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[12] Y. Li, A. Terrell, and J. M. Patel. WHAM: A high-throughput sequence alignment method. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*, SIGMOD '11, pages 445–456, New York, NY, USA, 2011. ACM.

[13] M. Massie, F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher, A. D. Joseph, and D. A. Patterson. ADAM: Genomics formats and processing patterns for cloud scale computing. Technical report, UCB/EECS-2013-207, EECS Department, University of California, Berkeley, 2013.

[14] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella,

D. Altshuler, S. Gabriel, M. Daly, et al. The Genome Analysis Toolkit: a mapreduce framework for analyzing next-generation DNA sequencing data. *Genome research*, 20(9):1297–1303, 2010.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[16] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan. Computational solutions to large-scale data management and analysis. *Nature Reviews Genetics*, 11(9):647–657, 2010.

[17] M. C. Schatz. CloudBurst: highly sensitive read mapping with mapreduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[18] E. R. Sparks, A. Talwalkar, V. Smith, J. Kottalam, X. Pan, J. Gonzalez, M. J. Franklin, M. I. Jordan, and T. Kraska. MLI: An API for distributed machine learning. In *2013 IEEE 13th International Conference on Data Mining (ICDM' 13)*, pages 1187–1192. IEEE, 2013.

[19] L. D. Stein et al. The case for cloud computing in genome informatics. *Genome Biology*, 11(5):207, 2010.

[20] A. Talwalkar, J. Liptrap, J. Newcomb, C. Hartl, J. Terhorst, K. Curtis, M. Bresler, Y. S. Song, M. I. Jordan, and D. Patterson. SMASH: A benchmarking toolkit for human genome variant calling. *Bioinformatics*, page btu345, 2014.

[21] M. J. Turk, B. D. Smith, J. S. Oishi, S. Skory, S. W. Skillman, T. Abel, and M. L. Norman. yt: A multi-code analysis toolkit for astrophysical simulation data. *The Astrophysical Journal Supplement Series*, 192(1):9, 2011.

[22] G. Wilson, D. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, et al. Best practices for scientific computing. *PLoS biology*, 12(1):e1001745, 2014.

[23] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica. GraphX: A resilient distributed graph system on Spark. In *First International Workshop on Graph Data Management Experiences and Systems*, page 2. ACM, 2013.

[24] 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 (HotCloud '10)*, page 10, 2010.

[25] H. Zimmermann. OSI reference model–the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.