Design Principles for Modular and Scalable Scientific Analysis Systems

ABSTRACT

Revolutions in data-acquisition are drastically changing how science conducts experiments. For example, the advent of "next-generation" sequencing technologies has led to a 10,000× decrease, which has led to exponential growth in the total volume of genome sequence data and improvements in our understanding of genetics. This broader trend has lead to the rise of "data science" and *data-driven discovery*, and many of these analysis problems are amenable to extant analytics systems.

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. 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. We also include a discussion of further optimizations we made to improve data management system performance.

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

[7]

1. Data science is a growing trend in both academia and industry
   (a) Driven by dramatic improvements in acquisition systems (e.g., sequencing, mass spectrometry, MRI systems)
   (b) Also driven by rise of statistical systems which are easy to use for non-experts (e.g., Scikit-learn [9], MLI [11])
2. Computing is becoming a dominant cost for science
3. We need efficient ways to construct these data processing systems
   (a) Efficiency is both computational cost and development cost
   (b) An efficient system should be fast
   (c) An efficient system should be able to support a full range of analyses
   (d) An efficient system shouldn't reinvent the wheel
4. Network stack achieves a similar goal:
   (a) Can swap out layers to tailor implementation
5. Contributions of this work:
   - Provide principles for the design of scientific analysis systems
   - Implemented fast genomic system
   - Implemented coordinate plane joins
   - Efficient lookup from block stores

2. BACKGROUND

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

1. MapReduce-based workflows
   (a) In CS, development of MapReduce → Hadoop → Spark
   (b) Equivalent systems in bioinformatics → GATK [8]
   (c) Hadoop-based genomics tools [10, 5]
   (d) Use of Spark for neuroscience
2. Database driven systems

   (a) SciDB [2]
   (b) GQL [4, 1]
3. Storage layers
   (a) CRAM [3]
   (b) YT [12]

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.

3.2 Layering

Discussion of Figure 1.

Specifically, we need to:

1. Compare across "big data" (i.e., COTS) and HPC settings
2. Show how current systems fit into the stack model, and how our proposed stack is different
3. Elucidate why it is more efficient to build systems that are decomposed as per our stack above (reference networking stack and protocol interchange)

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.

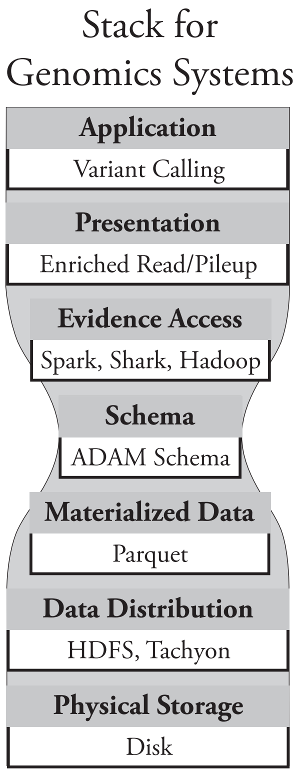

Figure 1: A Stack Model for Scientific Analysis

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 [6] 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 [11, 13].

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] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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, pages 445–456, New York, NY, USA, 2011. ACM.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.